CHARACTERIZATION OF UAV PERFORMANCE AND DEVELOPMENT OF A FORMATION FLIGHT CONTROLLER FOR MULTIPLE SMALL UAVS

THESIS

Patrick A. McCarthy, Ensign, USN

AFIT/GAE/ENY/06-J08

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GAE/ENY/06-J08

CHARACTERIZATION OF UAV PERFORMANCE AND DEVELOPMENT OF A FORMATION FLIGHT CONTROLLER FOR MULTIPLE SMALL UAVS

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Aeronautical Engineering

Patrick A. McCarthy, BS

Ensign, USN

June 2006

AFIT/GAE/ENX/06-J08

CHARACTERIZATION OF UAV PERFORMANCE AND DEVELOPMENT OF A FORMATION FLIGHT CONTROLLER FOR MULTIPLE SMALL UAVS

Patrick A. McCarthy

Ensign, USN

Approved:

_____	_____
Paul A. Blue, Major, USAF (Chairman)	Date
_____	_____
Dr. John F. Raquet (Member)	Date
_____	_____
Dr. David R. Jacques (Member)	Date

Abstract

The Air Force Institute of Technology's (AFIT) Advanced Navigation Technology (ANT) Center has recently delved into the research topic of small Unmanned Aerial Vehicles (UAV). One area of particular interest is using multiple small UAVs cooperatively to improve mission efficiency, as well as perform missions that couldn't be performed using vehicles independently. However, many of these missions require that the UAVs operate in close proximity with each other. This research lays the foundation required to use the ANT Center's UAVs for multi-vehicle missions (e.g. cooperatively) by accomplishing two major goals. First, it develops test procedures that can be used to characterize the tracking performance of a small UAV being controlled by a waypoint guided autopilot. This defines the size of the safety zones that must be maintained around each vehicle to ensure no collisions, assuming no, as yet unspecified, collision avoidance algorithm is being implemented. Secondly, a formation flight algorithm is developed that can be used to guide UAVs relative to each other using a waypoint guided autopilot. This is done by dynamically changing the waypoints. Such an approach gives a "wrap-around" method of cooperatively controlling UAVs that can only be guided waypoint-to-waypoint. For both components of this research, tests were conducted using a hardware-in-the-loop (HITL) simulation before validating through flight testing. This report, along with legacy documentation and procedures, furthers the UAV test bed at AFIT and establishes methods for simulating, visualizing, and flight testing multiple UAVs during formation/cooperative flight.

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Major Paul Blue, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated. Also, thanks goes to Dr. John Raquet, who manages the lab whose equipment this thesis made use of. This thesis could not have been accomplished without the help and support of the entire UAV team at AFIT. I will forever be indebted to Athan Waldron for his tireless effort in maintaining the aircraft and autopilot systems, as well as enduring countless hours of questions as I familiarized myself with the autopilots. John McNees, our RC aircraft expert and pilot, guided us through the experimental flight tests. This document will forever be linked to the other ANT Lab Ensigns, Joseph Dugan and Brent Robinson, for their brainstorming and problem-solving sessions, without which this thesis would never have been finished. I would also like to thank my sponsor, the Air Force Research Laboratory, Air Vehicles Directorate and the rest of the members of the Advanced Navigation Technology Laboratory at AFIT for both the support and latitude provided to me in this endeavor. Finally, I would like to thank my shipmates for their help and support throughout this graduate education experience.

Patrick A. McCarthy

Table of Contents

Page

Page

List of Figures

Page

Page

Page

Page

List of Tables

CHARACTERIZATION OF UAV PERFORMANCE AND DEVELOPMENT OF A FORMATION FLIGHT CONTROLLER FOR MULTIPLE SMALL UAVS

I. Introduction

1.1 – Motivation

Autonomous formation flight of unmanned aerial vehicles (UAVs) is a high priority within the Armed Forces and aerospace industry. The applications of UAVs are both numerous and valuable, especially for Department of Defense (DoD) objectives. UAVs of all sizes are being used in modern warfare because they remove the danger to human pilots from military operations as well as offer capabilities beyond those of piloted aircraft. Their possible uses include convoy protection, intelligence, surveillance and reconnaissance (ISR), autonomous combat operations and rapid over-the-hill surveillance for ground troops, to mention just a few.

Small UAVs in particular have become a recent research thrust for the DoD as well as the aerospace industry for many of the missions named above (Jodeh, 2005). With increasing scrutiny on the discrimination of military warfighting tactics, the necessity for precision operations by small UAVs in small, possibly urban battlefields becomes obvious. A small, inexpensive UAV capable of providing rapid intelligence to troops in the field would provide invaluable information currently out of reach by intelligence aircraft flying thousands of feet overhead.

Inexpensive, small UAVs also provide researchers the opportunity to investigate a number of autonomous flight applications without having to run costly wind-tunnel experiments or full-scale flight tests. There are a number of commercial-off-the-shelf

(COTS) UAVs available today for just a few hundred dollars, as well as COTS autopilot systems for just a few thousand dollars. These autopilot systems often come with hardware in the loop (HITL) simulation software, which offers the opportunity to verify performance before flight testing. Autopilots often are grouped with the UAV itself in the emerging term, Unmanned Aerial System (UAS), which will be used throughout this thesis when referring to both the UAV and the autopilot.

While UASs have been studied in-depth over the years on full-scale aircraft using expensive, proprietary technology (Osteroos, 2005), autonomous flight using small, inexpensive, commercial-off-the-shelf (COTS), waypoint-guided autopilots is a relatively new area of research. There is very little performance information available regarding the ability of these autopilots to accurately control small UAVs, let alone during formation or cooperative flight. There are just a few research test beds around the country that have been studying small UAVs over the past few years.

The Air Force Institute of Technology (AFIT) has recently established a UAS test bed at its Advanced Navigation Technology (ANT) Center, and is pursuing many research thrusts regarding these small UASs. Current topics of research include obstacle and collision avoidance, automated refueling, wind-correction, and formation flight. Prior to this thesis, Jodeh established the research platform by modeling, simulating, and performing initial flight tests (Jodeh, 2006). This research is being done concurrently with other AFIT MS Theses on boundary warning systems and synthetic vision (Dugan, 2006) and wind correction (Robinson, 2006). These theses are the first steps in setting up a research test bed at AFIT that will have greater research capabilities in the future.

The University of Pennsylvania developed and validated a hybrid model and experimentally controlled multiple UASs in 2004. Their hybrid model goes beyond the high-level autopilot controller and incorporates mode-switching logic using code developed in-house (Bayraktar et al., 2004).

Since 1997, The Massachusetts Institute of Technology's (MIT) Aerospace Controls Lab has published an extensive list of papers on various UAS topics in such areas as formation flying and carrier-phase differential GPS relative navigation, task assignment and path planning for UASs and spacecraft, and robust, nonlinear, and adaptive control. They have gone beyond the mere waypoint guided autopilots and developed their own optimized formation flight controllers and installed onboard computers so that the aircraft are wirelessly networked, and demonstrated these capabilities through experimental flight tests (Tin, 2004; King, 2004).

The University of California Berkeley's Center for the Cooperative Control of Unmanned Vehicles and the California Institute of Technology worked with both fixed and flocking formations for convoy protection and obstacle avoidance (Ryan et al., 2004).

There are several other UAS research test beds at universities, industrial research centers, and government laboratories across the country, including one at the Air Force Research Laboratory's Air Vehicles Directorate, with which the ANT Center at AFIT is collaborating.

Despite the number of groups currently working in the research area of small UASs, there are relatively few groups that have successfully flown multiple aircraft in

autonomous formation flight. According to Cloud Cap Technologies, the manufacturer of the Piccolo II autopilot used by most of the research groups, successful formation flight was first accomplished by the UCLA group in conjunction with Advanced Ceramics Research (ACR) in 2003, and has since been accomplished by the MIT group in 2004 (Cloud Cap, 2006). Multiple UASs were flown in 2005 by NASA and again in 2005 by Lew Aerospace using a specially developed operator interface (Cloud Cap, 2006). Their ground station was developed by Northrop Grumman and supports multi-vehicle operation over a single communications channel without any Piccolo II source code modification (Cloud Cap, 2006).

Because one of the goals of this thesis is to develop methods for formation flight that might extend to autopilot systems other than the Piccolo II, modification of the source codes of the autopilot or Operator Interface systems will be avoided. Instead, the Communications Software Development Kit (SDK), provided by the autopilot manufacturer, will be used to develop a user interface that interacts with the avionics. The formation flight algorithm developed in this thesis will be developed in as robust a method as possible, so that it might be applied to different waypoint-guided autopilots as well.

1.2 – Problem Statement

The goal of this research is to further the development of the UAV test bed at the Air Force Institute of Technology's ANT Center by characterizing the performance capabilities of small UASs using waypoint-guided autopilots and developing a formation

flight algorithm for such a UAS. The study of small UASs is relatively new, and there is little information available regarding their performance. The problem is complicated by the fact that a given UAS's performance significantly varies under different configurations, wind conditions, communications delays, and UAS dynamics. A detailed characterization of performance capabilities will determine the precision that can be achieved during formation flight and other multi-vehicle operations, such as cooperative control. This thesis addresses some of the preliminary issues that must be considered in order to successfully implement autonomous formation/cooperative flight of UASs. Once the performance limits have been established, an autonomous, waypoint-guided formation flight algorithm will be developed and verified through HITL testing and then validated through experimental flight testing.

1.3 – Research Objectives

- Develop flight test procedures that will determine the guidance and control characteristics of the UAS under various configurations.
- Determine the precision of formation flight that can be expected using particular UAS configurations.
- Develop a user interface that will interact with a waypoint-guided UAS and automatically update waypoints to position a trailing aircraft based on its relative position from a lead aircraft.
- Demonstrate the precision prescribed by the previously determined performance envelope using HITL simulation.

- Establish performance limits due to communications limitations for various airspeeds and configurations.
- Demonstrate the waypoint guided formation flight control through flight testing using a simulated lead aircraft with a follower UAS flying autonomously in formation.

1.4 – Significance of Research

The significance of this research is that it furthers the work done by Jodeh in 2006 in establishing the UAV test bed at the Air Force Institute of Technology's ANT Center, and provides a foundation for future formation and cooperative flight testing. Specifically, it provides a systematic procedure that can be used to characterize the precision with which a UAS can be maneuvered. This is essential information when operating multiple vehicles in close proximity of each other. This research also provides a formation control algorithm that can be used to operate waypoint guided UASs in formation or cooperatively; thus providing a retrofit for vehicles that can only be controlled waypoint-to-waypoint.

Furthermore, this thesis marks the first application of the SDK with the aircraft at AFIT, and will provide future students with a valuable guide to interacting with the avionics through the user interface developed at AFIT. This thesis develops procedures for flying multiple vehicles during HITL simulations with visualization as well as for experimental flight tests. These established procedures will save future researchers countless hours in familiarizing themselves with the autopilot system and allow them to instead focus on research. While complex formation flight controllers have been

developed by several other research groups using their own code, there is no published information available on using the SDK to write a simple formation flight controller by updating waypoints. This thesis will provide future research groups the opportunity to take the first step in autonomous formation flight without writing complex code.

1.5 – Methodology

The first step towards formation flight was setting up the hardware and software for multiple aircraft operation. Members of the UAV group at AFIT wrote procedures in 2005 for setting up Flight Gear, a common freeware flight visualization tool, for multiple vehicles (Vaglienti et al., 2005). The Operator Interface software provided with the Piccolo II autopilot allows for the simultaneous operation of up to 10 vehicles. The simulator was set up on a separate but networked computer for interaction with the operator interface and ground station. The intial setup was performed by Jodeh and the ANT Center UAV group in 2005 (Jodeh, 2006).

The Communications SDK was used to develop a formation flight controller implemented in C++. The code takes the position data of the lead aircraft, transforms it from Latitude, Longitude, Altitude (LLA) coordinates to East, North, Up (ENU) coordinates, and sends a new waypoint command to the trail aircraft based on a user-defined formation. It also adjusts the airspeed of the trail aircraft based on its distance behind the desired waypoint. For example, the trail flies much faster than the lead when it is a greater distance away, in an effort to catch up. When the trail aircraft gets closer to the lead aircraft, its commanded airspeed is set to more closely match the lead's. Because the communication of the ground station to the airborne autopilot is limited to a

speed of 1 Hz, the trail aircraft cannot get too close to the lead or it will pass the waypoint before a new waypoint is commanded. Therefore, there is a limitation on the formation separation based on airspeed.

Simulations were then run to establish performance limits on a single aircraft. Jodeh's thesis involved tweaking gains to improve autopilot performance, so the gains set by that research will be used here (Joseh, 2006). Any performance limits established during this research will be specific to this particular set of gains, which have not been optimized. Improvements to these gains could be done in future research, which could in turn improve the performance capabilities of the aircraft. However, the procedures established in this research can be applied to quickly characterize performance capabilities under future sets of gains.

Simulations will determine how far an aircraft deviates from a flight path by determining how close the aircraft comes to "hitting" each waypoint, and that deviation will be assigned as a formation limitation. Formation flight will then be simulated using the formation flight controller to send waypoint, speed, and altitude commands to the trail aircraft. The speed of response to lead maneuvers will be measured by importing the telemetry and control data to a MATLAB program, and formation performance capabilities will be established.

1.6 – Assumptions/Limitations

There are safety considerations that limit the ability to simultaneously fly multiple aircraft at this time.

1.7 – Preview

Chapter II examines the equipment used for the HITL, flight-testing, and the implementation of the formation flight controller. Chapter III walks through the actual development of the formation flight controller. Chapter IV covers the HITL simulations. Chapter V details the flight testing procedures that will govern single and multiple aircraft operation. Chapter VI compares the simulation results to the flight test results. Finally, Chapter VII provides conclusions and recommendations. All notable code and data is attached as Appendices.

II. Background

2.1 – Overview

The purpose of this chapter is to provide background information on all the equipment used during this research. Details of the aircraft and avionics will be included. Flight testing procedures will be discussed, as well as the basics of formation flight. Any other information necessary for a proper understanding of the thesis will be discussed. Finally, there will be a discussion on the collection and utilization of the Piccolo autopilot's telemetry and control data.

2.2 – Aircraft

2.2.1 – Airframe

The airframe chosen was the SIG Rascal 110. It was chosen for a number of reasons, including its large payload capacity, relatively stable performance characteristics, and its use across the country by other UAV research groups. Groups from the University of California Berkeley and the University of Colorado have each recently flown SIG Rascals in UAV related research applications mostly regarding vision-based navigation (Frew, et al, 2005). The Rascal 110 has a 110 inch wingspan, a high wing, and a tail wheel configuration. It was modified by AFIT to use a 50 oz fuel tank, which puts its endurance close to 2 hrs. The Rascal makes use of a hybrid airfoil that, according to the manufacturer, is a combination of the upper surface of an Eppler 193 and the lower surface of an Eppler 205, joined at the chord lines. Performance data about the aircraft and airfoil, as well as information regarding its weight, stability, and

balance, was presented by Jodeh (Jodeh, 2006). The aircraft comes disassembled but is Almost Ready to Fly (ARF). Once familiarized with the aircraft construction, the UAV group at AFIT is able to construct a fully capable Rascal in approximately 40 hours.

Figure 1 - SIG Rascal 110

2.2.2 – Engine and Propeller

The aircraft is powered by a FS-120S III four-cycle engine manufactured by O.S. Engines. It comes equipped with a diaphragm fuel pump, matching carburetor, and built-in pressure regulator, and outputs 2.1 horsepower at 12,000 rpm. The engine displaces 1.218 cubic inches (20 cc) (Engine Manual, 2000). A 16 x 8 propeller from APC was fitted to the engine. This combination of engine and propeller is capable of pulling the Rascal 110 faster than 60 knots under fair conditions. The engine is shown in Figure 2 (Engine Manual, 2000) and the propeller in Figure 3.

Figure 2 - FS-120S III Engine

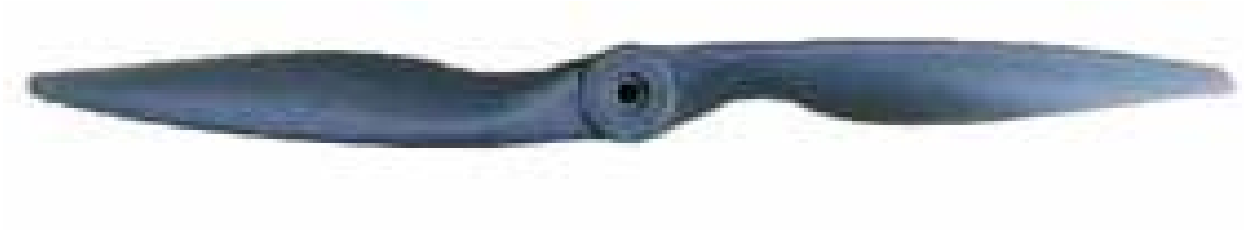

Figure 3 - 16 x 8 APC Propeller

2.3 – Avionics

The Avionics system has several major components, including the Piccolo II autopilot by Cloud Cap Technologies, the basic R/C system, servo actuators, and the Fail Safe Control Relay.

2.3.1 – Piccolo II Autopilot

The Piccolo II Autopilot is available for commercial purchase from Cloud Cap Technologies. It provides added functionality and flexibility from the original Piccolo I and PiccoloPlus, and can be used on both fixed-wing aircraft and helicopters. It has been incorporated on aircraft ranging from 3 lbs all the way up to nearly 1500 lbs. Its widespread use throughout the UAV research field can partially be attributed to its small size. The airborne component measures 5.25 inches deep by 2.5 inches high by 2.0 inches wide and weighs just over half of a pound. The Piccolo II is backwards compatible with previous Piccolo autopilots. The entire autopilot system consists of the airborne avionics, Ground Station Interface, HITL simulator, software, and a manual control box.

The airborne component, referred to as the Piccolo II, is shown in Figure 4 (Vaglienti et al., 2005). It incorporates a Motorola MPC555 microcontroller that processes inputs through a Reduced Instruction Set Computer (RISC). It provides 40MHz PowerPC operation, according to its manufacturers (Vaglienti et al., 2005). The Piccolo II collects air data through a dual ported mpxv50045 4kPa dynamic pressure sensor, an absolute ported mpx4115a barometric pressure sensor, and a board temperature sensor. It uses this data to calculate true airspeed (TAS), absolute altitude, and ambient air temperature. It makes use of three ADXRS300 gyros and dual two-axis ADXL210e accelerometers for rate and acceleration measurements (Vaglienti et al., 2005).

Figure 4 - Piccolo II Airborne Avionics Package

The Piccolo II estimates wind by calculating the difference between GPS Ground Speed and TAS each time the aircraft turns. Attitude and gyro bias are estimated by a Kalman filter, which uses the GPS-derived pseudo-attitude as the measurement correction (Vaglienti et al., 2005).

The Piccolo II datalink is built on a 1W 900MHz and a 1W 2.4GHz radio modem, and sends command and control, autopilot telemetry, payload data transfer functions, differential GPS corrections uplink, and pilot in the loop modes at rates up to 40Kbaud. The datalink architecture is designed in a way that allows a single operator interface to control multiple aircraft from a single ground station (Vaglienti et al., 2005). The GPS receiver used by the Piccolo II is the μBlox TIM LP, an update from previous systems. The μBlox has an output of 4Hz. The avionics system controls up to 10 servo outputs and two independent Controller Area Network (CAN) buses. The pulse width

modulation (PWM) outputs can be operated in two basic modes: 5-channel mode and 10-channel mode. The system at AFIT will operate on a 5-channel mode. Figure 5 shows the avionics relationships in the form of a block diagram (Vaglienti et al., 2005).

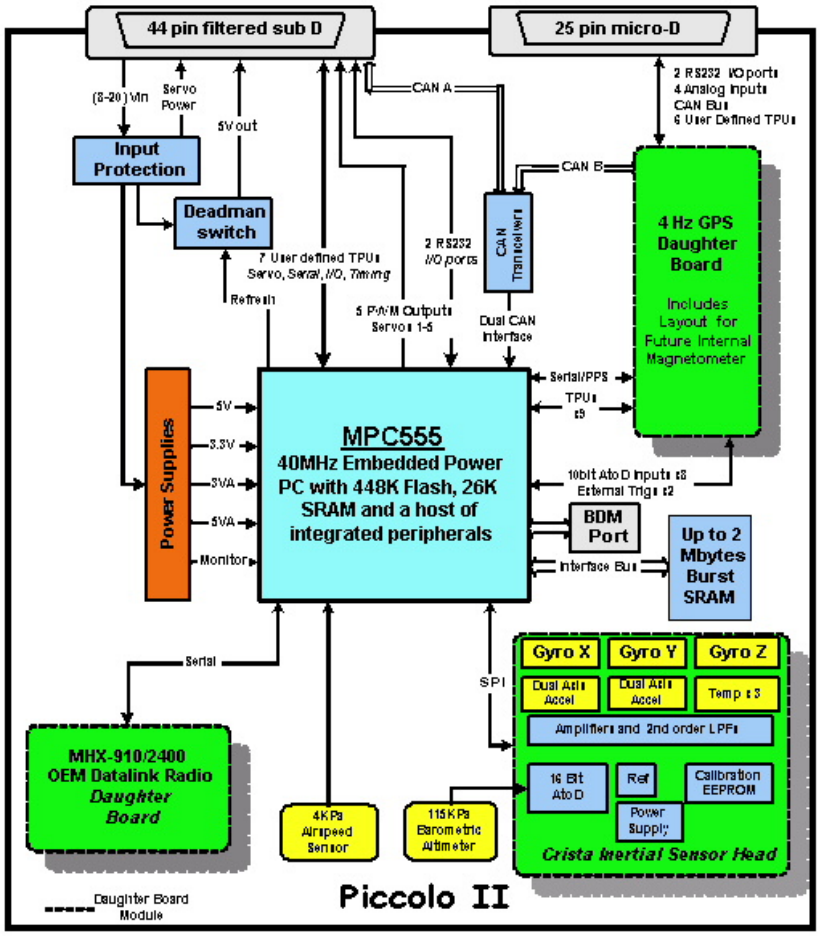

Figure 5 - Piccolo II Block Diagram

The ground equipment includes the Ground Station, a UHF antenna, a GPS antenna, a laptop, and an R/C control box, and is shown in Figure 6. For HITL simulations, a second computer is generally needed to run the Simulator, and is connected to the Ground Station through a USB to Controller Area Network (CAN)

module. The R/C control box is only for pilot-in-the-loop operation, and can be disregarded for autonomous flight. It is necessary as a safety precaution, so that operation can be switched from autonomous flight to piloted flight in emergencies or during testing. The laptop is used to run the Operator Interface, provided by the manufacturer. It provides a user-interface for interacting with the avionics, and is capable of sending commands and updating gains in real-time. It also receives and logs all data received from the autopilot, including telemetry and control, equipment status, communications status, etc. An example of one of the windows of the Operator Interface, the one used to manipulate gains, is shown in

Figure 7.

Figure 6 - Ground Equipment Minus Laptop

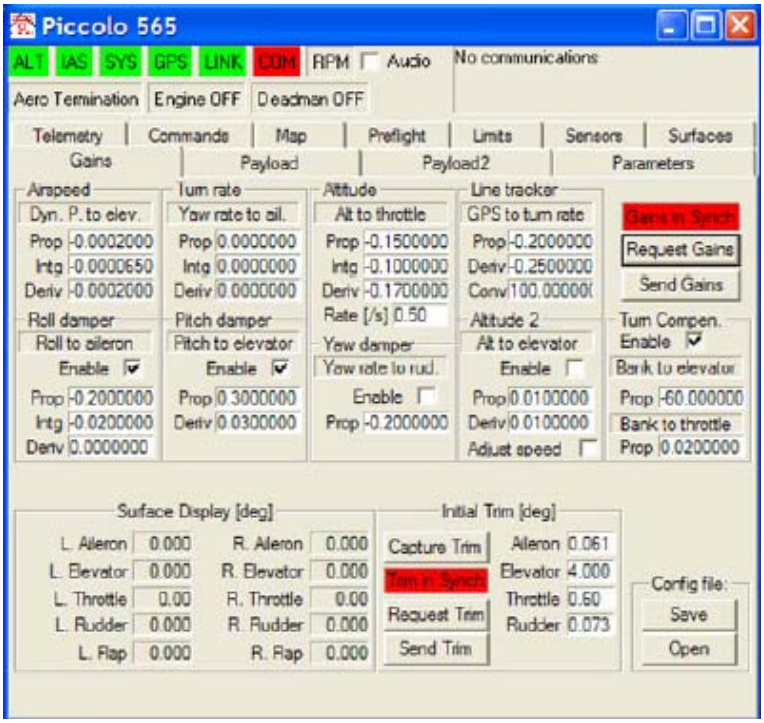

Figure 7 - Operator Interface

Finally, the Ground Station manages the communications between the laptop and the airborne autopilot. It also interfaces to the pilot-in-the-loop console and supervises all of the communications links when multiple networks are in operation. Communication between multiple networks and the Ground Station is shown in Figure 8.

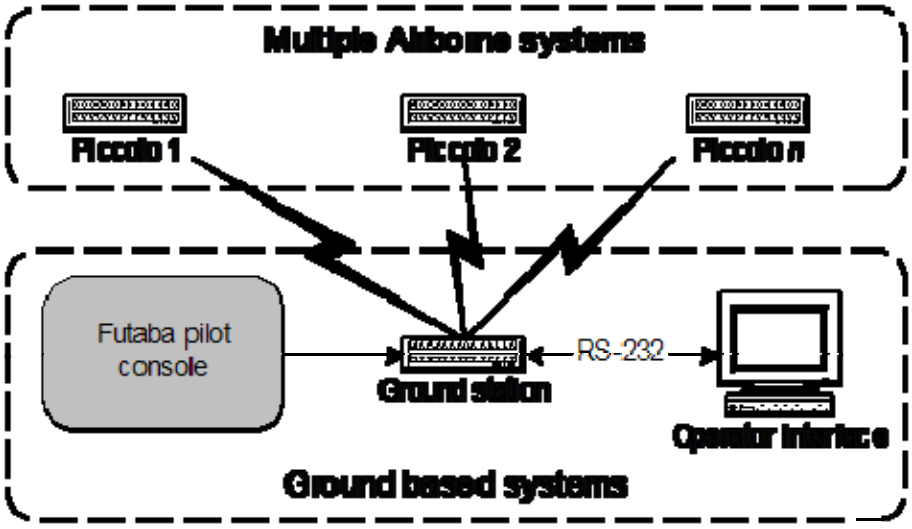

Figure 8 - Overview of Ground Station Communication Architecture

2.3.2 – Radio Control System

The Radio Control system consists of an 8 channel Futaba 9CAP/9CAF transmitter with an R149DP PCM 1024 receiver.

2.3.3 – Fail Safe Control Relay

As a safety precaution, the Air Force Research Labs Sensors Directorate (AFRL/SN) designed a Fail Safe Control Relay. Developed before this thesis was started, the Fail Safe Control Relay functions as an automatic switch between R/C transmitter and autopilot controller if a loss of communications occurs. In his thesis, Jodeh describes the system in more detail (Jodeh, 2006). A diagram showing the interaction between the avionics systems and power paths is shown in Figure 9.

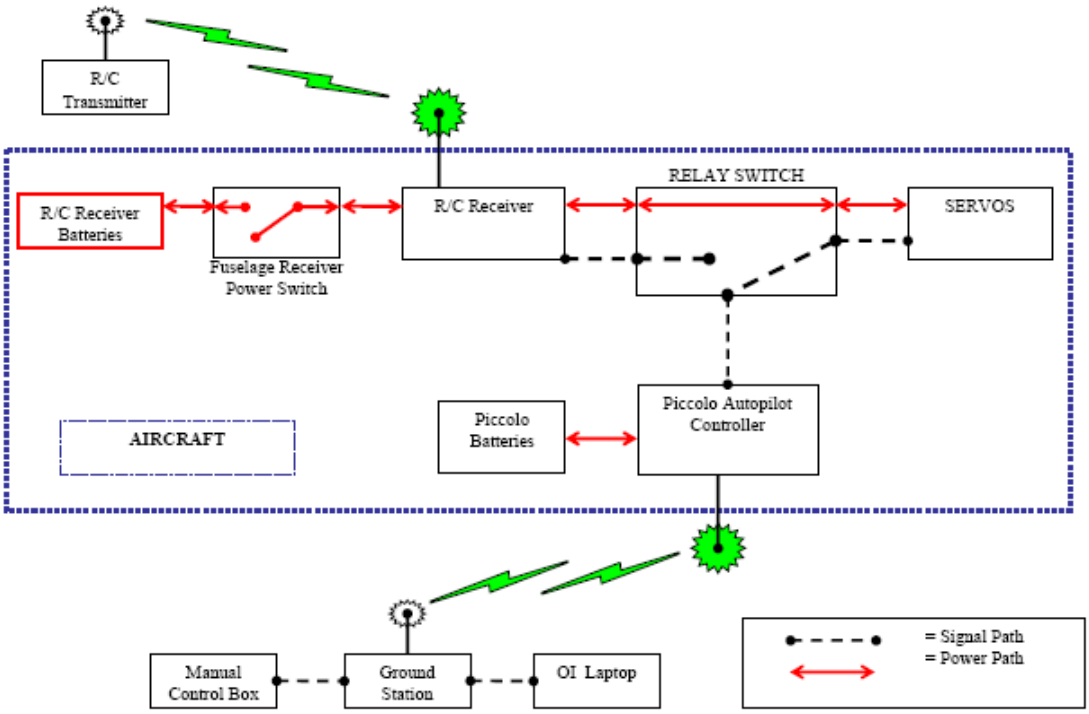

Figure 9 - Avionics Communications

2.4 – Visualization

HITL simulations need not be visualized, but it is a valuable tool for mentally translating data into actual aircraft performance. The visualization software provided with the Piccolo II is the Flight Gear flight simulator, an open source program that is capable of receiving UDP network packets from a separate PC. Cloud Cap Technology's User Manual describes the setup of Flight Gear in detail in the Hardware in the Loop Simulator for Piccolo Avionics section. The setup of a single visualization PC is shown in Figure 10.

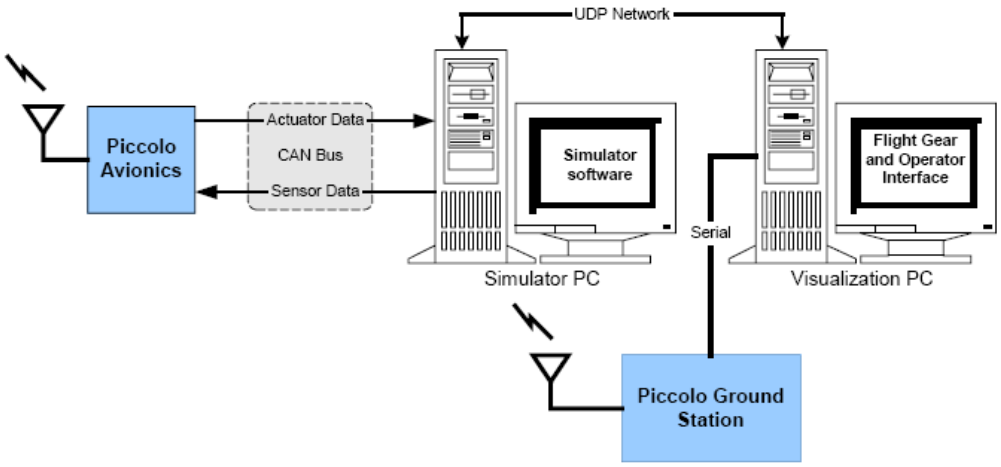

Figure 10 - HITL Simulation with Visualization

The setup for visualization of multiple aircraft is only slightly more complicated. On a local network, multiple simulators can be running off of the same Ground Station and Operator Interface. Each simulator simply has to send UDP packets to a different IP address, and Flight Gear has to be initialized with a few simple command line parameters (Vaglienti et al., 2005). A multiple aircraft visualization is shown in Figure 11.

Figure 11 - Multiple Aircraft Displayed in Flight Gear

2.5 – Basics of Formation Flight

2.5.1 – Coordinate Transformation

All GPS data received by the avionics comes in Latitude/Longitude/Altitude (LLA) coordinates, which makes a transformation into a simpler coordinate system necessary. East/North/Up (ENU) coordinates are based in Cartesian coordinates, which makes them much easier to work with and allows for a better performance evaluation. The limits of formation flight will be determined by the ability to track waypoints under

varying conditions; so the physical distance between an aircraft's targeted position and actual position must be known.

ENU is a commonly used inertial coordinate system for aircraft, and is similar to North, East, Down (NED), which varies only in a sign change on the altitude coordinate. Converting from LLA to ENU is not possible through any single rotation matrix. First, the LLA coordinates must be converted to Earth-Center-Earth-Fixed (ECEF) coordinates before they can be converted to ENU. ECEF coordinates are cartesian coordinates that define a three-dimensional position with respect to the center of mass of the Earth, as illustrated in Figure 12 (Baleri, 2006). The conversion is done through a series of steps which will now be described.

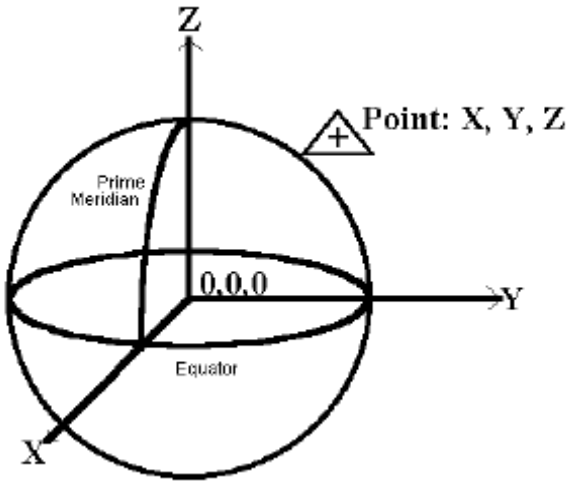

Figure 12 – ECEF Coordinate Reference Frame

A base point must first be established in ECEF coordinates. For this, an arbitrary reference point in LLA coordinates was chosen near the flight test area. This coordinate

was converted to ECEF coordinates (x,y,z) according to the following equations (Baleri, 2006).

$$e_1^2 = \frac{a^2 - b^2}{a^2} \tag{1}$$

$$N = \frac{a}{\sqrt{1 - e_1^2 \sin^2(\gamma)}} \tag{2}$$

$$x = (N + h)\cos(\gamma)\cos(\lambda) \tag{3}$$

$$y = (N + h)\cos(\gamma)\sin(\lambda) \tag{4}$$

$$z = (\frac{b^2}{a^2} N + h)\sin(\gamma) \tag{5}$$

where

a = semi-major axis of Earth = 6378137m

b = semi-minor axis of Earth = 6356752.31424518m

γ = latitude of reference point

λ = longitude of reference point

h = height above ellipsoid

N = Radius of Curvature

The numerical values used for each axis are based on the commonly used WGS-84 geodetic datum (Baleri, 2006). The value for h an altitude above ground level.

Next, quaternions were used to convert from ECEF to ENU coordinates. While it is possible to convert between ECEF and ENU through Euler angle rotations, these angles are based on trigonometric functions that face singularity problems at certain

values. Quaternions, on the other hand, involve a four parameter system not based on trigonometric functions, thereby eliminating any singularity problems. A more detailed discussion of quaternions can be found in detail in (Stevens, 2003).

The quaternion conversion between ECEF and ENU coordinates is commonly encountered and is available open source. Figure 13 and Figure 14 show the 2-dimensional flight path of the same simulation in LLA and ENU coordinates, respectively. Figure 15 and Figure 16 show the same simulation from a 3-D perspective.

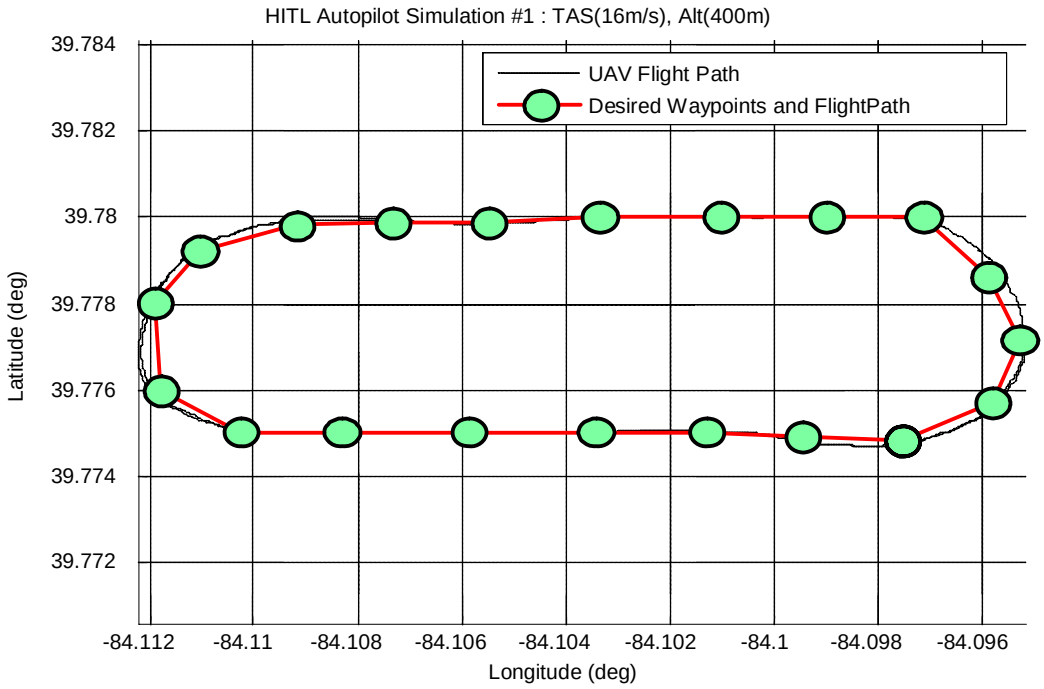

Figure 13 - Simulation in 2-D LLA Coordinates

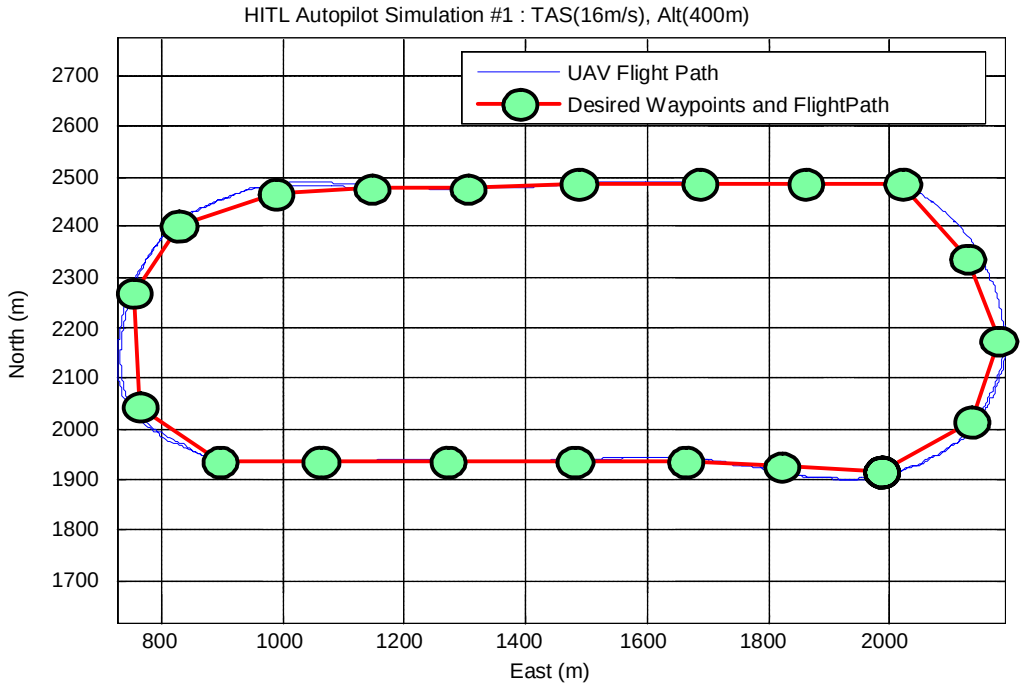

Figure 14 - Simulation in 2-D ENU Coordinates

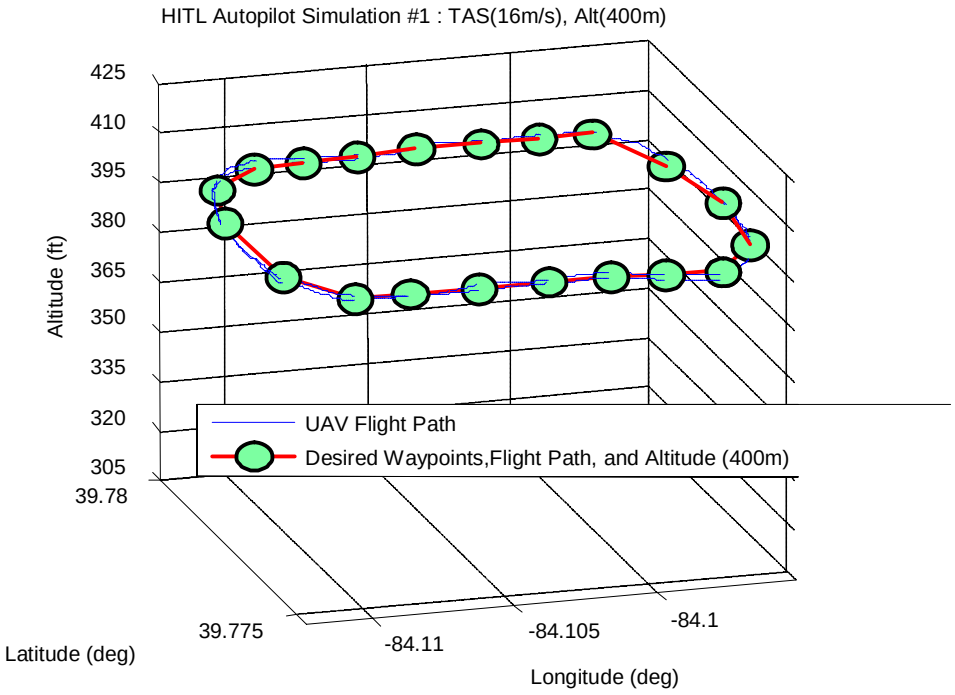

Figure 15 - Simulation in 3-D LLA Coordinates

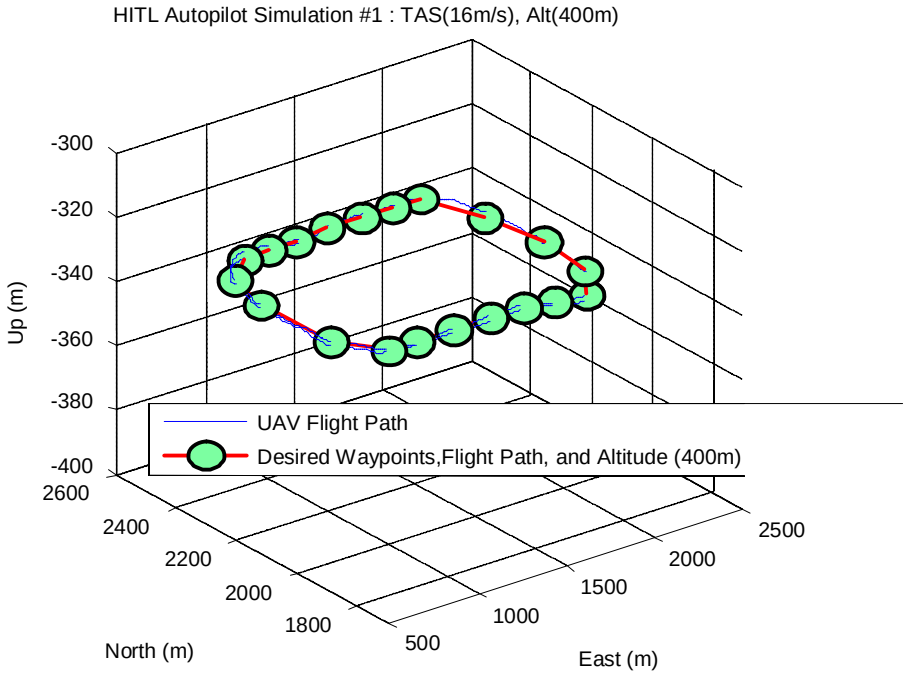

Figure 16 - Simulation in 3-D ENU Coordinates

2.5.2 – Formation Flight

Formation flight, as discussed in this thesis, consists of a lead aircraft and at least one follower whose commands are determined by maneuvers of the lead. The Piccolo II is a waypoint-guided autopilot system; so the simplest form of formation flight involves sending continuously updating waypoints to the follower aircraft based on the lead aircraft's position. The trail aircraft's airspeed will also have to be adjusted based on its relative position to the lead.

The relative position of the trail aircraft will be a function of three variables, the 2-D separation in the North-East plane, R, the altitude separation in the Up plane, Z, and the angle formed between the reverse of the lead aircraft's heading and the trail aircraft's position, θ. Figure 17 shows formation flight using these variables. Because this thesis will employ waypoint-guided navigation rather than sending surface deflection commands, the Euler angles and surface deflections of the lead aircraft do not need to be known by the trail aircraft. Chapter IV will discuss the insertion of waypoints and adjusting of commands to the trail aircraft in more detail.

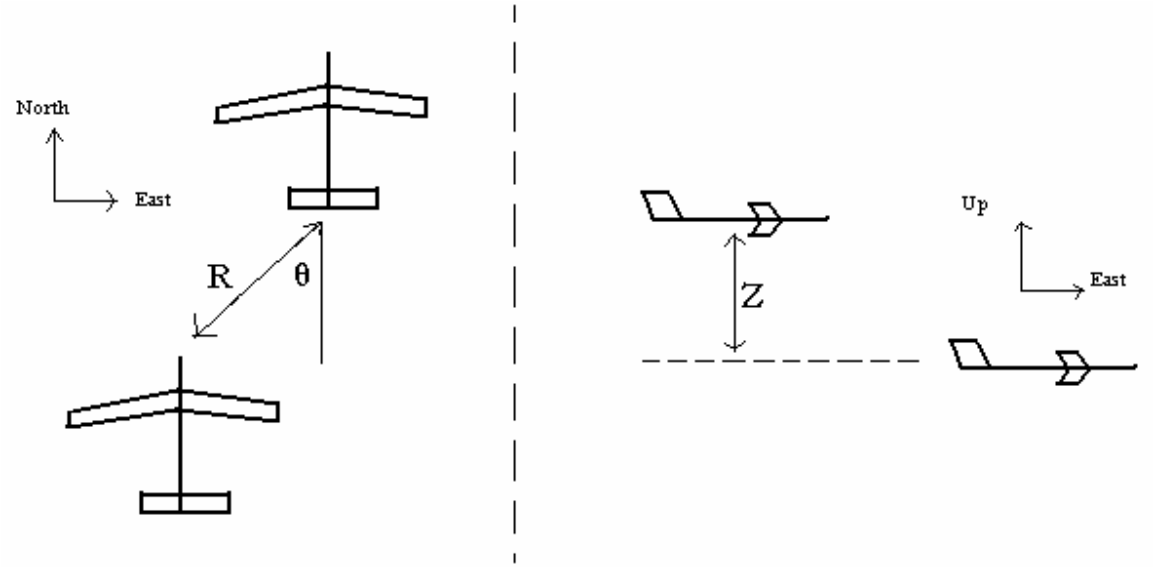

Figure 17 - Formation Frame of Reference

2.6 – Flight Testing

Flight testing was conducted at WPAFB during June 2006. The following section will provide a brief introduction to the flight testing process.

2.6.1 – Overview of Flight Testing

Much of the work to set up flight testing operations for the ANT Center's UAV test bed was performed by Jodeh (Jodeh, 2006). The aircraft's status as Almost-Ready-to-Fly led to relatively few gain-tuning flight tests before autonomous flight testing was possible. Jodeh's thesis provides a more detailed discussion of the flight testing process (Jodeh, 2006).

2.6.2 – Flight Test Range and Equipment

Flight tests were conducted over a large enclosed runway area located on Area B at Wright-Patterson AFB. The trapezoidal-shaped area that was approved by the Safety Review Board is shown in Figure 18.

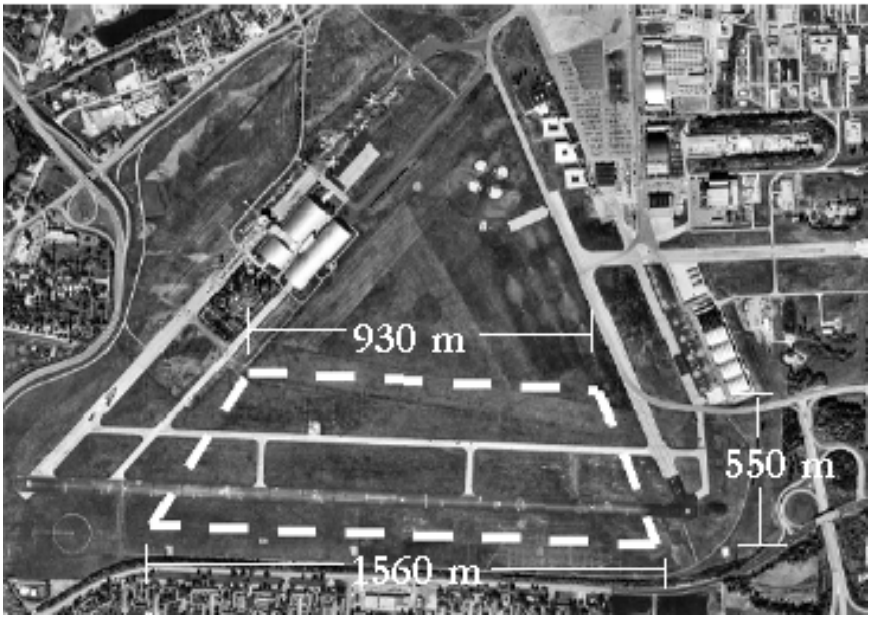

Figure 18 - WPAFB Area B Flight Test Range

2.6.3 – WPAFB Flight Testing Criteria

There were a number of concerns for flight testing on WPAFB. The base's proximity to major roads and residential areas, as well as its normal aircraft operations, led to a stringent set of rules for flight testing that had to be approved by three review boards, including a Configuration Control Board (CCB), a Safety Review Board (SRB), and a Technical Review Board (TRB). A number of guidelines were set based on the recommendations of those boards, advice from AFRL/SN flight test team, expert R/C

pilots, and Cloud Cap Technologies (Lozier et al., 2005). Table 1 outlines some of those guidelines (Jodeh, 2006).

Table 1 - Flight Testing Criteria

Winds Less than 30 mph
Temperature Greater than 40°F
Visibility Greater than 3 Miles
Cloud Ceiling Minimum 500 ft AGL
Airspace Ceiling Maximum 400 ft AGL
GPS Satellites 6 or more visible
Radio Frequency Interference Check
Safety Equipment and First Aid Kit
Pitch, Roll, and Yaw Rate Gyro Operations
Static and Dynamic Pressure Port Operation
WPAFB Control Tower Notification

2.7 – Collection and Management of Telemetry and Control Data

The Piccolo II autopilot logs telemetry and control data through the Operator Interface. It logs 70 total parameters at sampling rates of either 20Hz or 1Hz. The different sampling rates, termed “request fast” for 20Hz and “request slow” for 1Hz, are necessary because of the incredible size each log can reach. With a 20Hz data collection

rate, files can grow to over 30Mb in just 30 minutes of simulation or flight. In a one hour flight sampling at 20Hz, the logs could include telemetry and control data for up to 72,000 samplings, or over half a million data points. Dealing with data files this large becomes exceedingly difficult using normal data-manipulation software such as Microsoft Excel or MATLAB. Therefore, the 20Hz setting was only used during maneuvers that required studying. In between maneuvers, the rate was switched to 1Hz.

Once a flight was completed and the Operator Interface was closed, the logs could be read into Microsoft Excel. The log files listed the data in columns below a text header. Excel was able to import the data, which was delimited by spaces. Certain parameters could be thrown out because they were irrelevant to this research, including onboard temperatures, battery voltages, GPS signal strengths. Of the 70 parameters, the only values of importance to this thesis are listed in Table 2.

Table 2 - Important Logging Parameters

Time (ms)
Hours
Min
Sec
Latitude
Longitude
Altitude
TAS
Target_Waypoint_Index

This simplified log file was then saved and imported to a MATLAB workspace, which created column vectors for each parameter. The data was imported into Excel before MATLAB because MATLAB is less able to deal with certain non-numerical parameters. Those columns were erased in Excel, prior to importing the data to MATLAB. Once in MATLAB, these new column vectors of parameters could be manipulated and particular portions of them could be graphed based on the timing of aircraft maneuvers. The indices of important times were recorded during testing to properly plot important data corresponding to particular maneuvers.

Collecting data for formation flight was slightly different. The Operator Interface does not record formation information, such as separations and bearings; so the user interface for the formation flight controller was modified to output certain parameters directly to a text file at the same sampling rate as the Operator Interface. All flight formation data was sent to this file as well as the timestamp corresponding to each line of data. These text files were then read into Excel and MATLAB in the method described above.

2.8 – Chapter Conclusion

The hardware involved in this project was almost entirely COTS, with the exception of the Fail Safe Control Relay. The flight testing procedures and data collection tools have been established. The LLA to ENU coordinate transformation facilitated the evaluation of the aircraft's flight (e.g. the accuracy of the guidance and control). The simulation tools provided and procedures established will provide a

valuable, safe way to run experiments without the risk of an actual flight test. Chapter III will detail the methodology behind implementing the Software Development Kit.

III. Waypoint Guided Formation Flight Controller

3.1 – Chapter Overview

This chapter will first provide an overview of the Piccolo II communications and the capabilities of the Software Development Kit (SDK). Next, the formation flight controller developed using the SDK will be described. Finally, there will be a discussion on the tuning of the formation flight controller based on simulations.

3.2 – Piccolo II Communications Overview

The MHX 900Mhz or 2.4GHz ISM band radio form a wireless link between the airborne avionics and the ground station. It operates on a frequency hopping network that enables the ground station to communicate to multiple networks at once. Running simultaneous networks brings up the need for some sort of flow control, which the Piccolo II manages in a "round robin" fashion, where it looks for data from each network one after the other (Vaglienti et al., 2005).

Each network sends packets over the datalink that is destined for a particular stream, which is identified by the first byte in the packet. The communications for the ground station are just slightly more complicated because it must multiplex stream data because of its interaction with multiple networks at once. A more detailed explanation of the Piccolo II Communications can be found in the Communications for the Piccolo Avionics section of the Piccolo User's Manual. Figure 19 diagrams the communications of the autopilot system (Vaglienti et al., 2005).

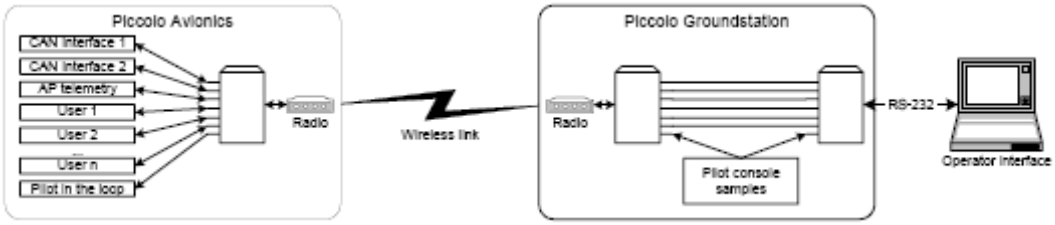

Figure 19 – Piccolo II Communications Facilities

3.3 – Software Development Kit Overview

The Communications Software Development Kit (SDK) provides a method for interacting with the avionics without altering the Piccolo II source code. While the SDK calls for the development of any software in C++, it is a hybrid mix of C and C++ because the avionics and ground station software were written entirely in C (Vaglienti et al., 2005).

The SDK creates a simple Win32 Application and provides a Communications Manager (CommManager) Class , which serves as the basis for interacting with the avionics through the formation flight controller. The CommManager provides a number of functions for sending and receiving packets from the avionics. The user is tasked with writing functions that alter any commands the avionics is receiving and sending those new commands back to the avionics.

3.4 – Formation Flight Controller Development

Using the SDK, a controller was developed that allows for formation flight using a waypoint-insertion/airspeed adjustment/altitude adjustment method.

3.4.1 – Waypoint Insertion

To determine the proper position for waypoint insertion, the position and orientation of the lead aircraft must be known. The ground station receives all positional data in LLA coordinates; so they are first transformed to ENU coordinates using the method described in Chapter 2.5.1. The new desired waypoint position for the trail aircraft is then calculated based on a desired two-dimensional radial separation, a bearing from the lead aircraft, and an altitude separation. The bearing towards the lead aircraft is based on the angle between the desired position and the reverse of the lead aircraft's heading. For example, if a desired position is 50m radial separation, 0m altitude separation and 45° bearing, the desired waypoint would be inserted as shown in Figure 20. The formation flight controller developed has a number of predefined formations, shown in Figure 21 and described in Table 3. The formation flight controller developed allows a user to change a variety of inputs, including altering the formation in real-time.

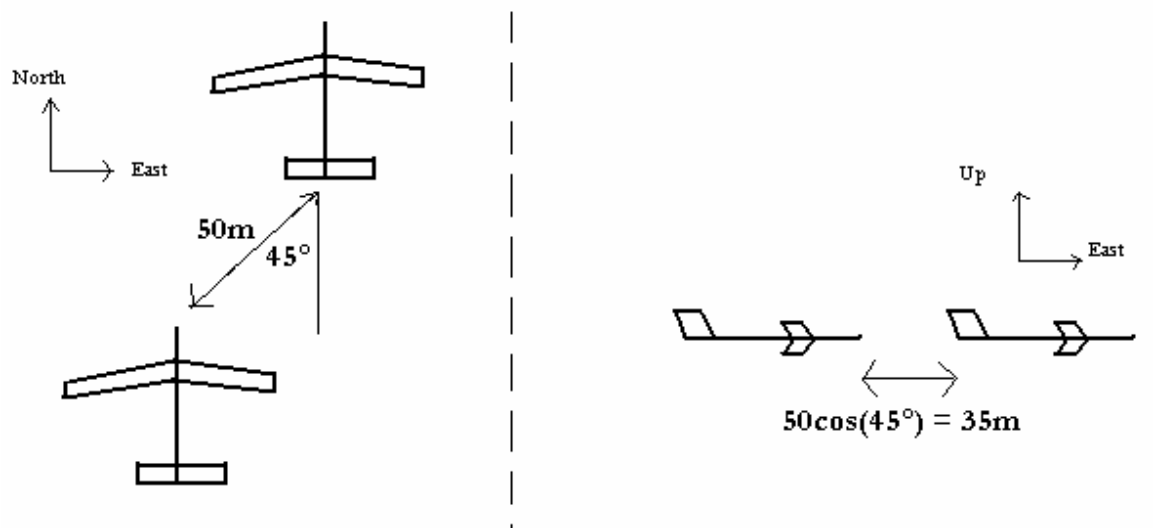

Figure 20 - Formation Example

Table 3 - Formations

Formation Type	2-D Radial Separation (m)	Bearing (degrees)	Altitude Separation (m)
Line	50	0	0
V	50	45	0
Above	0	0	50
Wing	50	90	0

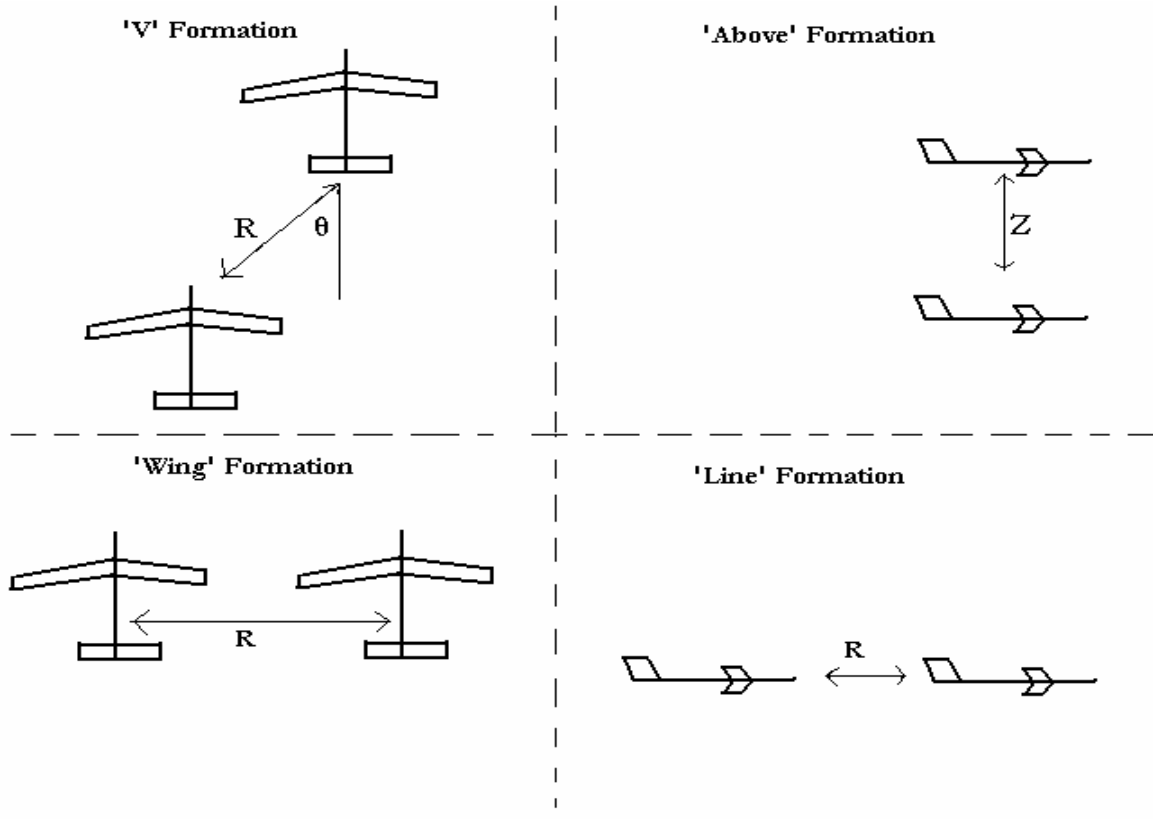

Figure 21 - Predefined Formations

Once the ENU coordinates of the desired trail aircraft's waypoint position are calculated based on the lead aircraft's position and the desired formation data, it is simply a matter of converting back to LLA coordinates and inserting the waypoint by broadcasting it to the trail's Piccolo II by way of the Ground Station. The C++ source code for the formation flight controller developed can be found in Appendix B.

3.4.2 – Maintaining Desired Separation

In waypoint control mode, the Piccolo II attempts to guide the aircraft to the next waypoint. However, the waypoints are not "time-stamped", so the autopilot flies the aircraft at a specified airspeed to the waypoint without regard to when it arrives at the

waypoint. It quickly becomes obvious that a trail aircraft will have to be sent dynamically updating airspeed commands in order to maintain the desired separation.

Designing a speed controller to maintain the desired separation between the lead and trail aircraft is itself a challenging problem that could be accomplished many different ways, including using linear models and linear control theory or using nonlinear models and nonlinear control theory. However, it is essential to account for the communication delays that exist between the Ground Station, where the formation flight controller is implemented, and the airborne UASs. Since the aircraft models available did not account for these delays, and since the constraints of time made it impossible to generate them during this thesis, an ad-hoc design method was used to develop a speed controller using the HITL simulation, which includes all of the communication delays. The speed controller developed changes the commanded airspeed of the trail aircraft based on its distance from the desired waypoint. The relative position between the lead and trail aircraft is calculated by converting their GPS positional data to ENU coordinates. The trail aircraft's position is then compared to the position of the desired waypoint based on formation information. Based on the distance between the trail aircraft and its desired position, a new airspeed, based off of the lead's airspeed, is commanded. More separation between the trail and its desired waypoint means a higher target airspeed is sent to the trail. Without any communication delay, the trail aircraft could ideally fly nearly on top of the updating waypoint. However, this is impossible because there is a significant time delay between sending a waypoint command to the trail aircraft and the aircraft actually maneuvering towards that waypoint. This time

delay required the formation flight controller to incorporate a built-in separation, which could not be surpassed.

The first method for tuning these commanded airspeeds was basically guess-and-check. This method will be referred to in this thesis as the "stop-and-go" method of gain adjustment, named so because it has jumps in commanded airspeeds as the trail crosses certain separation thresholds. Hundreds of simulations were run with the lead aircraft flying at 16m/s while visualizing both aircraft to fine-tune the ratios of commanded airspeeds at particular distances. Table 4 shows a breakdown of the separations and commanded airspeeds that were finally settled on. The separation distances presented in this table were determined through a combination of a study of the single aircraft performance limits and countless formation flight simulations, and are valid for a lead airspeed of 16m/s. The 40m separation defined in Table 4 as the lower limit of formation flight incorporates the time delay, the results of which will be shown in Chapter VI. This built-in separation will have to be altered to account for varying a lead aircraft's airspeed.

Table 4 – Stop-and-Go Trail Airspeed Adjustment

Separation Between Desired and Actual Position (m)	Bearing Condition	Ratio of Commanded Trail Airspeed to Lead Airspeed
Separation > 300	None	1.33
300 > Separation > 100	None	1.22
100 > Separation > 50	Deviation>20°	0.84
100 > Separation > 50	Deviation<20°	1.09
50 > Separation > 40	Deviation>20°	0.84
50 > Separation > 40	Deviation<20°	1.048
40 > Separation > 0	None	0.77

It must be noted that a trail aircraft could be significantly off its desired bearing to the lead aircraft; so a check was inserted to correct for that, as shown in the second column of Table 4. Figure 22 shows the necessity for this correction. A trail aircraft whose formation called for it to fly directly behind the lead aircraft could fly alongside it in a wing formation instead, and it would be constantly increasing speed as long as it maintained more than enough separation. It becomes necessary to slow the trail aircraft down if it gets too far off-bearing. Because this is most likely a problem in the scenarios when separation is less than 100m, only these scenarios will be adjusted. It can be seen that the trail was commanded slower airspeeds when more than 20 degrees from its desired bearing.

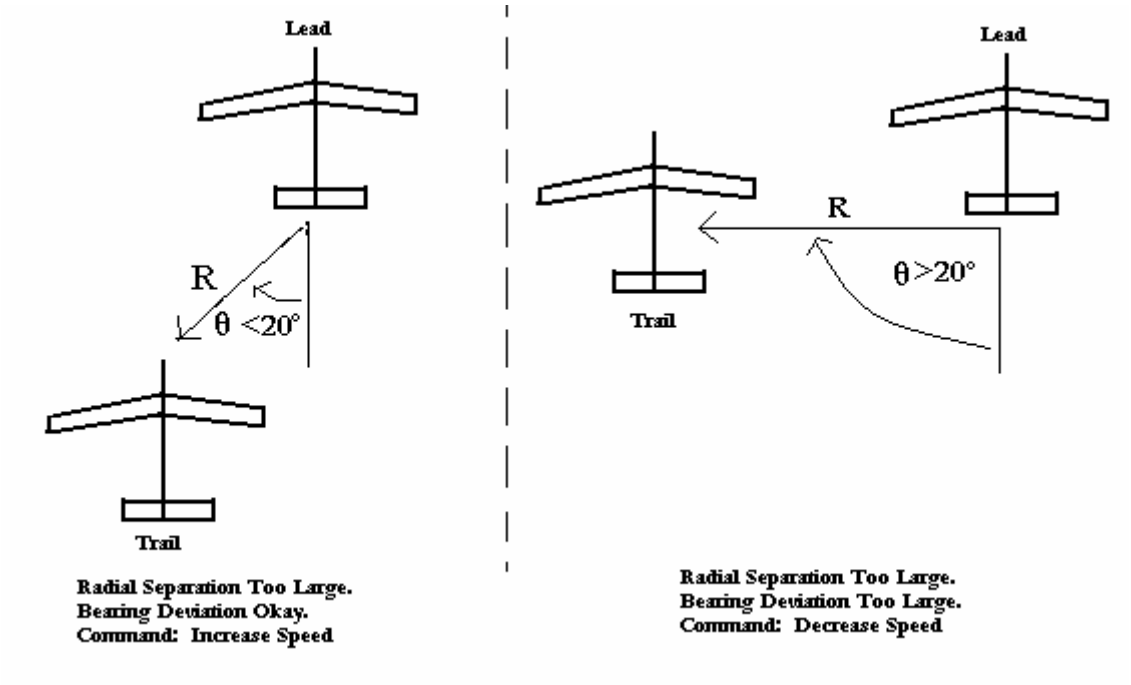

Figure 22 - Formation Controller Bearing Condition

This ad-hoc controller, while adequate at its task, could be greatly improved by fitting a regression curve to it. Through a regression analysis, shown in Figure 23, it was determined that a logarithmic controller would be ideal for this particular system. A logarithmic controller is capable of providing superior control during both close formation and when the separation is several orders of magnitude greater than desired. The particular control law developed is according to the following equation.

$$V_{Trail} = V_{Lead}[k1 + k2\ln(S+1)] \tag{6}$$

where

V_{Lead} = TAS of lead aircraft

V_{trail} = Commanded TAS of trail aircraft

S = Distance between trail aircraft and desired waypoint

k1 = Gain 1 > 0

k2 = Gain 2 > 0

This equation was developed through a regression analysis and shifted slightly (the "S+1" term) to ensure a positive velocity command is always given. This control law should provide superior performance because there are no "jumps" in airspeed across the separation conditions. It also allows for the trail aircraft to match the lead's airspeed exactly instead of continuously speeding up or slowing down to maintain proper separation. This would greatly improve formation performance during steady level flight. However, the formation performance will strongly depend on the tuning of the gains. Low gains could lead to slow response, while high gains could cause the trail to overshoot its desired waypoint. The gains must be tuned properly in order to improve formation performance.

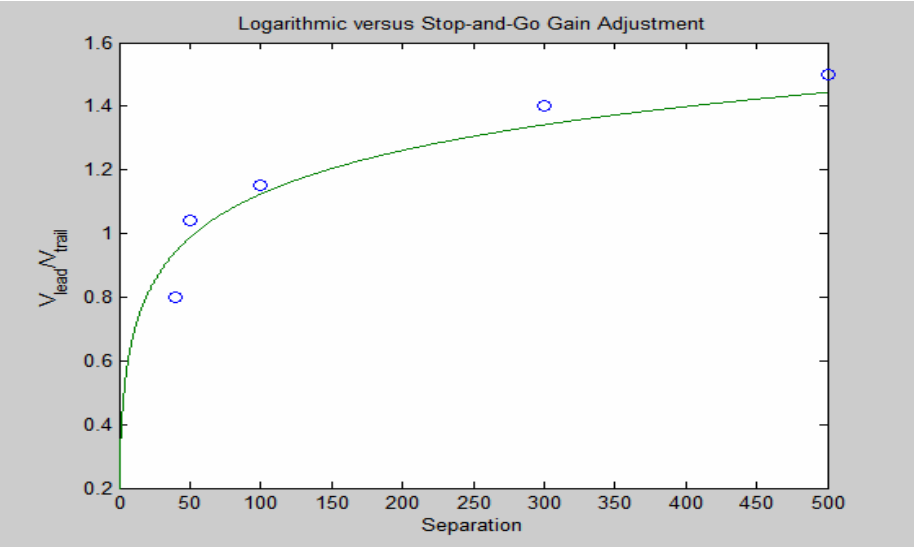

Figure 23 - Logarithmic Gain Fit

3.5 – Gain Tuning

This method of adjusting airspeed for formation flight is still not ideal, and the gains, k1 and k2, need to be fine-tuned for adequate performance. The gains were tuned by examining the telemetry and control data and through visualizing formation flight through Flight Gear. The gains that were settled on were a k1 of 0.2 and k2 of 0.2. Obviously these gains will perform differently at different airspeeds. When the lead's airspeed is high, the trail will have a more difficult time staying in formation. The effect of different airspeeds on formation capabilities will be studied in Chapter VI.

3.6 – Airspeed Adjustment at Varying Lead Airspeeds

For the most part, the methods of gain tuning described above were performed at a lead aircraft airspeed of 16m/s. The reasoning behind this will be evident when the results of the single aircraft performance evaluation are presented in Chapter VI. 16m/s will prove to be an optimal airspeed for both formation flight and single aircraft flight in the prescribed flight test area. However, the methods for extending the airspeed controller will be described here.

The separations predefined in the "stop-and-go" trail airspeed adjustment are largely dependent on the lead aircraft's airspeed. The minimum separation distance should be defined by the following relationship:

$$S_{min} = V_L \times t_d \tag{7}$$

where

S_{min} = Minimum Separation Distance

V_L = Lead Aircraft Airspeed

t_d = Time Delay

The time delay will be presented in Chapter VI, but the results of those simulations were necessary for tuning the formation flight controller. The time delay ended up being around 4 seconds, which would indicate a minimum separation at 16m/s of 64m. The initial minimum separation distance for a lead airspeed of 16m/s was set of 64, but it was discovered through HITL simulation with visualization that the formations could be even tighter than that. This could be due to a number of factors. First, the time delay might not always be as high as 4 seconds. Also, the trail aircraft rarely approaches the minimum separation. As the results of the formation flight simulations will show in Chapter VI, having a minimum separation distance actually yields an average formation separation of around 64m for a lead airspeed of 16m/s, a separation nearly identical to what the time delay predicts. To extend the minimum separation distances to other airspeeds, the separations will simply be multiplied by the ratio of 16m/s to the new lead airspeed. For example, for a lead airspeed of 24m/s, the 0 < Separation < 40 condition would change to 0 < Separation x (16/24) < 40. The same holds true for the logarithmic gain adjustment, whose equation will now be:

$$V_{Trail} = V_{Lead}[k1 + k2\ln(S[16/V_{Lead}]+1)] \tag{8}$$

The results of using this method of gain adjustment will not be studied in this thesis because of time constraints. The size of the time delay was not discovered until near the end of this thesis, and the necessity of using minimum separation distances as a function of time didn't become apparent until the very end of this research. Therefore, formation flight testing will be valid only for 16m/s, but the methods described above

should extend to varying airspeeds and provide better performance than the results shown in Chapter VI.

3.6 – Chapter Conclusions

The SDK provides a valuable tool for developing software to interact with the Piccolo II. The development of a formation flight controller and the methods for inserting waypoints and adjusting airspeeds to achieve formation flight were described above, and the results of both HITL simulations and flight tests using this controller will be discussed in later chapters.

IV. Hardware-in-the-Loop Simulation

4.1 - Chapter Overview

This chapter will detail the development of HITL simulations for both single and multiple aircraft. The chosen flight plans and various operating conditions will be explained. The single aircraft performance evaluation will be discussed first, followed by the multiple aircraft formation flight simulations. The results of the simulations discussed in this Chapter are presented in Chapter VI.

4.2 – Single Aircraft Simulations for Performance Evaluation

4.2.1 – Waypoint Following

Determining how closely an aircraft can track a series of closely spaced waypoints is the ultimate goal of this performance evaluation, because that will determine how tight a formation can be achieved using a waypoint-guided autopilot. A series of conditions were selected to simulate various conditions the aircraft will face in actual flight tests. A single aircraft was assigned a number of flight plans and they were flown at varying speeds and gain settings to study the effect of speed on waypoint following. The logs were then analyzed in MATLAB and a code was developed to evaluate any deviation from the aircraft's desired waypoints.

The flight plans used to study the aircraft's deviation from desired position consisted of racetrack patterns of varying radii with a large number of closely spaced waypoints. A smaller, tighter racetrack more closely approximates the actual flight test

area, while a larger, looser racetrack provides a glimpse at optimal performance in a flight test area where space is not an issue. This may be useful for later research at AFIT. Another important factor in waypoint-following performance with the Piccolo II autopilot is the track convergence gain, defined by the manufacturer as the "track convergence parameter." Counter-intuitively, smaller track convergence parameters direct the aircraft to fly closer to the track between waypoints as possible. Larger track convergence parameters, on the other hand, send the aircraft to the next waypoint without regard to the track connecting it to the previous waypoint. Figure 24 illustrates the effect of the track convergence parameter. Table 5 summarizes the various flight conditions that were studied in these waypoint-following simulations.

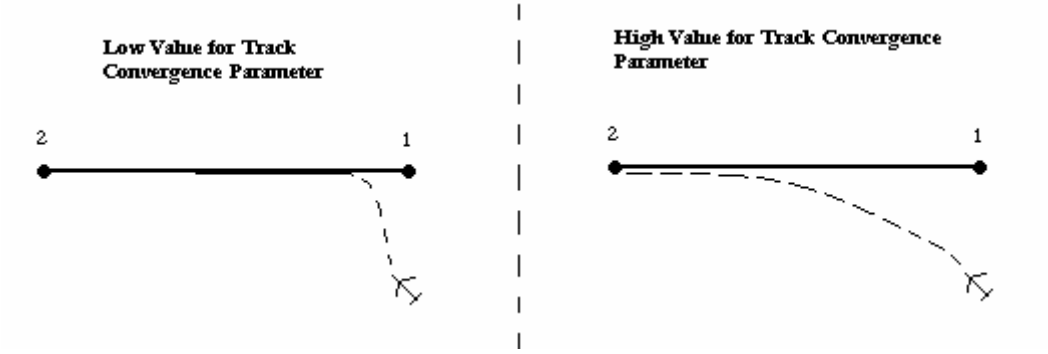

Figure 24 - Effect of Track Convergence Parameter

Table 5 - Single Aircraft Waypoint-Following Simulations

Simulation #	Race Track	Track Convergence	Airspeed (m/s)
1	Tight	250	16
2	Tight	250	25
3	Tight	50	25
4	Loose	250	25
5	Loose	250	16
6	Loose	50	16

4.2.2 – Turning Radius

The commanded airspeed is obviously a big factor in determining how tight an aircraft is capable of turning. The mathematics behind the turning radius of an aircraft are shown in the following equation (Rogers, 2001).

$$TR = \frac{V^2}{g \tan(\phi)} \tag{9}$$

where

TR = turning radius

V = airspeed

g = gravity

φ = bank angle

Therefore, the aircraft's minimum turning radius occurs at its maximum bank angle, and will change proportionally with the square of the TAS. Another important

thing to consider in studying the turning radius of an aircraft is the effect of turning on altitude. An aircraft will often lose significant altitude during tight turns, because its lift vector is no longer balancing its weight, as shown in Figure 25. The Piccolo II autopilot, however, includes compensators that maintain steady level flight when the gains are tuned properly; so maximum bank angle will most likely not be achieved. This will be shown through the results of several simulations in Chapter VII.

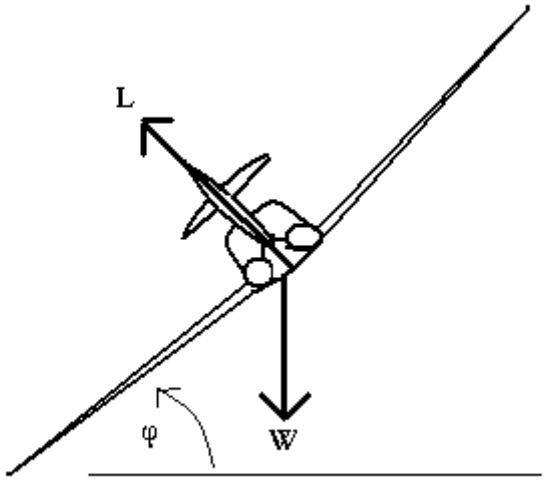

Figure 25 - Aircraft during Left Bank

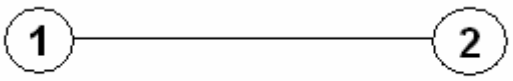

Figure 26 - Turning Radius Flight Plan

A number of simulations were run studying the effect of airspeed on turning radius and altitude. The flight path for these simulations consisted of just two waypoints, as shown in Figure 26. The Piccolo II's "preturn" setting was turned off for this

simulation, so the aircraft was directed to fly all the way to one waypoint before turning. The Piccolo II was commanded to hold a specific altitude while making the 180 degree turn. Varying track convergence parameters were studied, as this would prove to be a major factor in achieving better performance. The results of these tests will determine any limits on speed or turning radius that a lead aircraft might encounter during formation flight. Table 6 summarizes the various conditions that were studied during the turning radius simulations.

Table 6 - Single Aircraft Turning Radius Simulations

Simulation #	Track Convergence	Airspeed (m/s)
7	50	25
8	250	25
9	250	16
10	50	16

4.3 – Formation Flight Simulations

4.3.1 – Time Delays

There are a number of delays associated with a waypoint guided autopilot, especially during cooperative or formation flight. The trail aircraft is basing its waypoints off the lead aircraft's position, but the ground station only communicates with each airborne avionics box at 1Hz. There are a number of steps in the formation flight process that introduce time delays, which are outlined as follows:

- Ground Station receives lead aircraft telemetry and control data and sends to Operator Interface
- Formation flight controller reads telemetry and control data
- Formation flight controller computes necessary waypoint position and control data for trail aircraft
- Formation flight controller sends new commands to Operator Interface
- Operator Interface sends data to Ground Station, which is put in queue to be sent to the trail aircraft
- Trail aircraft receives commands
- Avionics processes commands and operates servos accordingly

It is obvious that this entire process will create a significant time delay between the lead aircraft performing a maneuver and the trail aircraft adjusting to stay in formation. The 1Hz communication rate of the ground station is particularly limiting, because the trail aircraft may not receive a command until sometimes 2 seconds after a lead maneuver. The magnitude of the overall delay will be determined through several simple simulations. The lead aircraft will be manually commanded increases in altitude and speed in separate simulations, and the lag between the lead's maneuver and the trail's maneuver will be analyzed. Table 7 summarizes the time delay simulations, and the results can be found in Chapter VI.

Table 7 - Formation Flight Time Delay Simulations

Simulation #	Altitude Change (m)	Airspeed Change (m/s)
11	+50	0
12	0	+9

4.3.2 – Varying Formations

Various formations were hard-coded into the formation flight controller but will not be analyzed in simulation due to time constraints. The only formation that will be analyzed in detail will be the "above" formation, as shown in Figure 21, with a trail aircraft flying 50m above its leader. This will provide an easy way to determine the minimum formation separation possible because there is no radial separation between the lead aircraft and the trail aircraft's commanded waypoint. The only problem that could arise with different formations would be on hard turns directly into the trail aircraft's flight path when the trail aircraft is flying in a "V" or "Wing" formation. This can be avoided if the results of the time delay and minimum turning radius simulations are used to set a minimum formation separation.

4.3.3 – Varying Airspeeds

The tightness of the formation will be largely dependent on airspeed. Therefore, the bulk of the formation flight simulations will be flown in the same racetrack pattern that will be used during flight tests on Area B of WPAFB. A single formation was flown in a much larger racetrack to determine the best formation performance the aircraft can hope to achieve in unlimited space. Table 8 summarizes the formation flight simulations.

Table 8 - Formation Flight Simulations

Simulation #	Flight Plan	Formation	Track Convergence	Airspeed (m/s)
13	Large Racetrack	Above	250	16
14	Area B Plan	Above	250	16
15	Area B Plan	Above	250	25
16	Area B Plan	Above	250	12
17	Area B Plan	Above	50	12
18	Area B Plan	Above	50	16

4.4 – Chapter Conclusions

Simulations were run to evaluate the performance of a single aircraft using the set of gains previously obtained at AFIT. A number of simulations were run to determine the waypoint-following capabilities and minimum turning radius while holding altitude. These limitations were used in the formation flight controller to develop the gains for formation flight. Formation flight was simulated using the HITL simulator and the formation flight controller. The delay between lead and trail aircraft maneuvers during formation flight was studied. Various airspeeds and gain settings were studied to determine the optimal settings for formation flight.

V. Flight Testing Procedures

5.1 – Chapter Overview

This chapter will describe the flight-testing of a single aircraft. It will serve as a continuation of the initial discussion on flight test procedures begun in Chapter II and described in great detail in Jodeh's thesis (Jodeh, 2006). Multiple aircraft will not actually be flown, but a simulated lead aircraft will fly and a trail aircraft will attempt to stay in formation. The setup of this simulated lead will be described, as well as the flight procedures and formation maneuvers of the aircraft. Unfortunately, circumstances beyond the control of this author forced flight testing to be canceled. It will be accomplished at a later date using the same procedures and maneuvers described in this thesis.

5.2 – Testing Issues and Limitations

As has been described several times, the small flight test area severely limits the type of tests that can be performed. Of the 10 simulations performed in the lab, only those performed on smaller flight plans with lower airspeeds will be attempted in experimental flight tests. Only the flight tests deemed crucial to validating conclusions drawn from the HITL simulations will be flown.

As discussed before, there are a number of issues regarding autonomous flight of multiple aircraft that become problematic. The SRB, TRB, and CCB that convened for Jodeh's thesis approved autonomous flight for a single aircraft, but the entire process will have to be repeated for multiple aircraft. This is not possible in the time allotted for this

thesis; so a lead aircraft will be simulated and the trail aircraft will fly in a “virtual” formation, with the simulated lead.

5.3 – Single Aircraft Flight Procedures and Maneuvers

The procedures for single aircraft flight testing were described in Chapter 2.6. First, all of the conditions established by the SRB, TRB, and CCB were met. The key members of the flight test team include the RC pilot, the operator manning the Operator Interface, a Safety Officer, and several spotters. The RC pilot guides the aircraft through takeoff and climbs to altitude, at which point the controls are switched to autonomous flight. The operator is capable of sending updated flight plans and gains in real-time to the autopilot. The telemetry and control data is recorded by selecting at a higher rate during specific maneuvers, and switched to the lower rate the rest of the time.

The most important flight testing maneuvers for the single aircraft research focus on two main issues: waypoint-following and determining the turning radius of an aircraft. The waypoint-following flight plan will consist of a number of closely spaced waypoints flown on the tight, Area B racetrack. Several laps will be flown at varying track convergences at 16m/s airspeed.

The turning-radius flight will be the same as Simulation #9 (Table 7), which consisted of an aircraft flying between two waypoints at 16m/s with a low track convergence parameter. The higher track convergence parameter simulations will not be validated through flight testing because this test hopes to determine the minimum turning radius, which is clearly accomplished through low track convergence parameters.

5.4 – Simulated Lead Aircraft Setup

As described above, multiple vehicles will not be flown at once. Instead, a lead aircraft will be simulated, with the formation flight controller sending commands to the trail aircraft based on the lead's simulated flight. To simulate a lead aircraft, all the HITL hardware and software had to be brought into the field. The simulator software and formation flight controller was set up on one laptop, and the Operator Interface was set up on another. A local area network (LAN) was established between the two laptops. The Operator Interface was set to recognize the airborne aircraft as the "Pilot" so that in case of emergency the Piccolo manual controller would take control of the actual aircraft rather than the simulated one. The simulation was then no different than running it in the lab as always, except that a single computer was being used to run a simulation and the formation flight controller rather than using separate computers for each task.

5.5 – Multiple Aircraft Flight Procedures and Maneuvers

The procedures for flight testing multiple vehicles are virtually identical to those for single aircraft flight testing. The team will consist of the RC pilot, an operator manning the Operator Interface, a Safety Officer, several spotters, and another operator manning the formation flight controller interface. The two operators will work together to send the aircraft on different flight plans and adjust the gains. The formation flight controller operator will toggle the formation flight on and off, as well as command new formations to the trail aircraft. He or she will also monitor the formation separations, to warn of any (simulated) collision.

As described before, a trail aircraft will be commanded waypoints, airspeeds, and altitudes based on the separation between it and a simulated lead aircraft. Two main flight plans will be used during this testing. They will be the Area B racetrack and a straight line plan similar to the turning radius plan used during single aircraft flight testing. The tests conducted using the straight line plan will demonstrate the formation time delays, similar to Simulations #11 and #12. The tests on the Area B flight plan will be flown using both methods of gain adjustment, as well as various track convergence parameters for the lead aircraft. All plans will be flown at 16 m/s, as that has been established as the optimal speed for this flight testing area.

5.6 – Chapter Conclusions

The flight testing procedures and maneuvers were described in this Chapter as a continuation of the overview presented in Chapter II. The flight tests that will be conducted will validate those HITL simulations that show important results. The results of the single aircraft flight testing will determine the capabilities of formation flight. For multiple vehicles, the flight testing procedures and maneuvers outlined in this chapter validate the functionality of the formation flight algorithm. The flight tests performed show the important limitations that need to be considered when defining formation separations and angles. Experimental flight testing was not possible in the time allotted for this thesis, but all procedures will be followed by future researchers to validate the simulated results of this thesis. Chapter VI will present the specific optimal formation

settings based on the goals of a particular flight, as well as the formation separations that could be achieved using the existing set of gains and hardware.

VI. Results and Analysis

6.1 – Chapter Overview

This chapter will discuss the results of all simulations and flight tests and compare the two. It will provide reasoning behind gain tuning and other autopilot settings.

6.2 – Single Aircraft HITL Simulation Results

The results of the simulations described in Chapter IV are detailed below. Figure 27 through Figure 29 show the tight racetrack pattern flown at 16m/s with a track convergence of 250. These racetrack patterns with many waypoints were flown to show the ability to track closely spaced waypoints, therefore the closest point of approach was calculated in MATLAB and the average deviation between the aircraft's simulated position and desired waypoint position is shown in Figure 29. Altitude and airspeed as functions of time are shown in Figure 28 to show that there is minimal altitude drop and variation of airspeed in this first simulation, which does not approach the limits of aircraft performance.

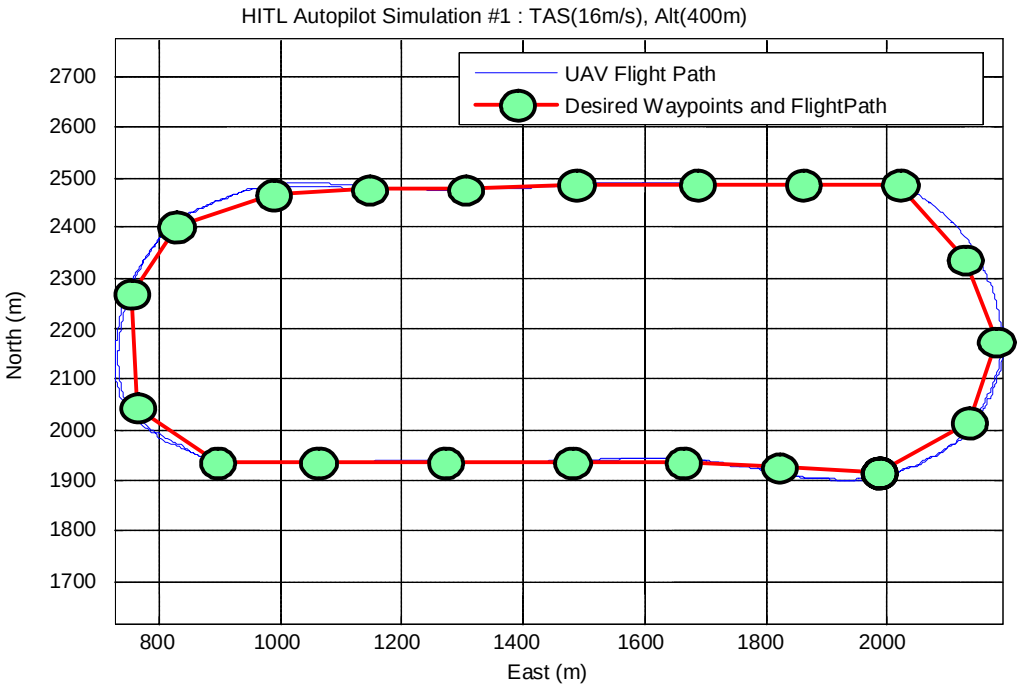

Figure 27 - Simulation #1 Flight Path

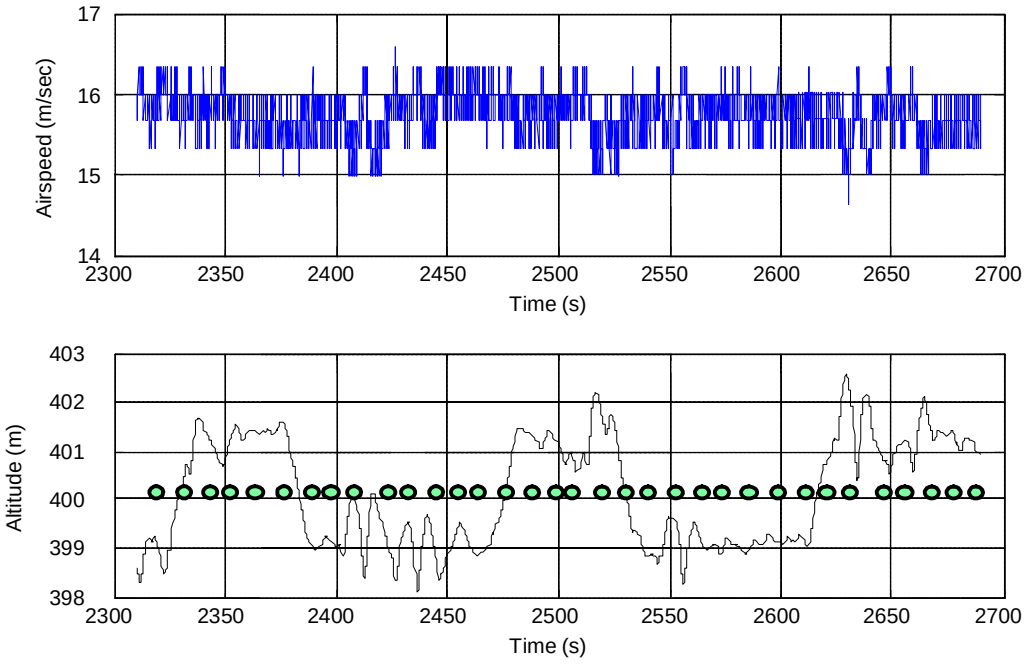

Figure 28 - Simulation #1 Altitude and Airspeed versus Time

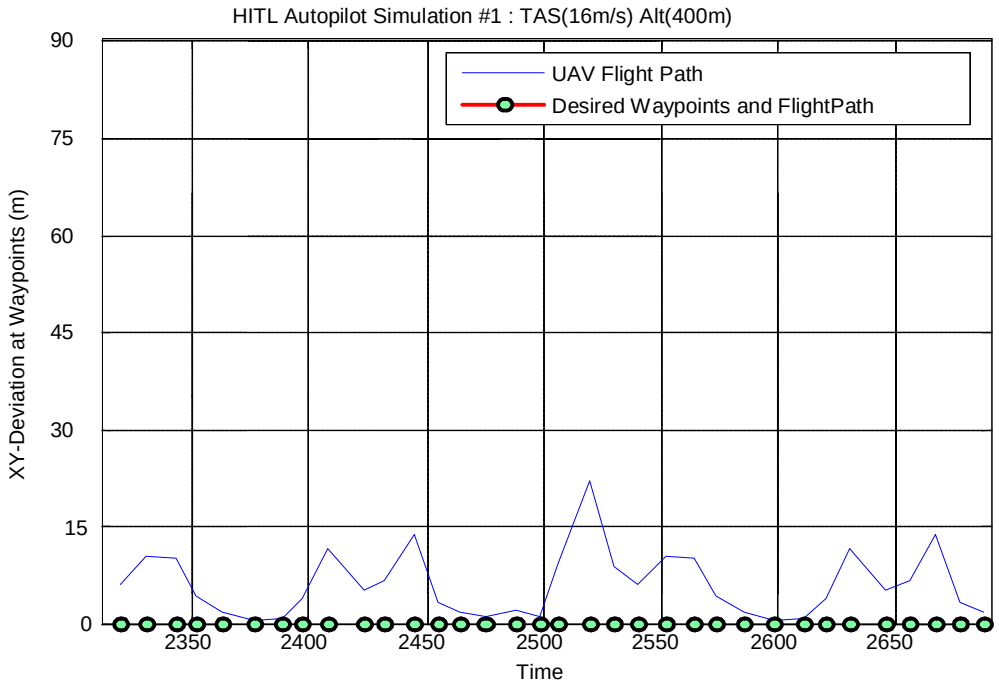

Figure 29 - Simulation #1 2-D Deviation at Waypoints

As Figure 29 shows, maximum deviation at a waypoint is just 22m. Figure 30 and Figure 31 show the path of a simulation flying a tight racetrack with track convergence 250 and 25m/s airspeed.

A number of other racetrack patterns with varying track convergences, airspeeds, and track radii were flown and can be found in Appendix A. Only those simulations that show performance limitations of the aircraft will be discussed in this section.

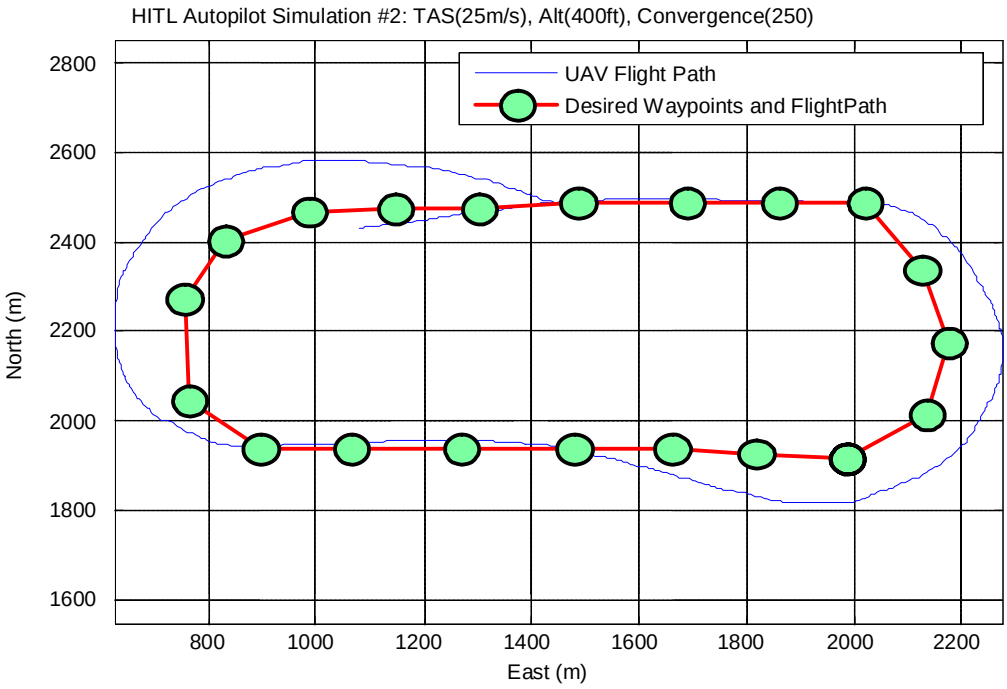

Figure 30 - Simulation #2 Flight Path

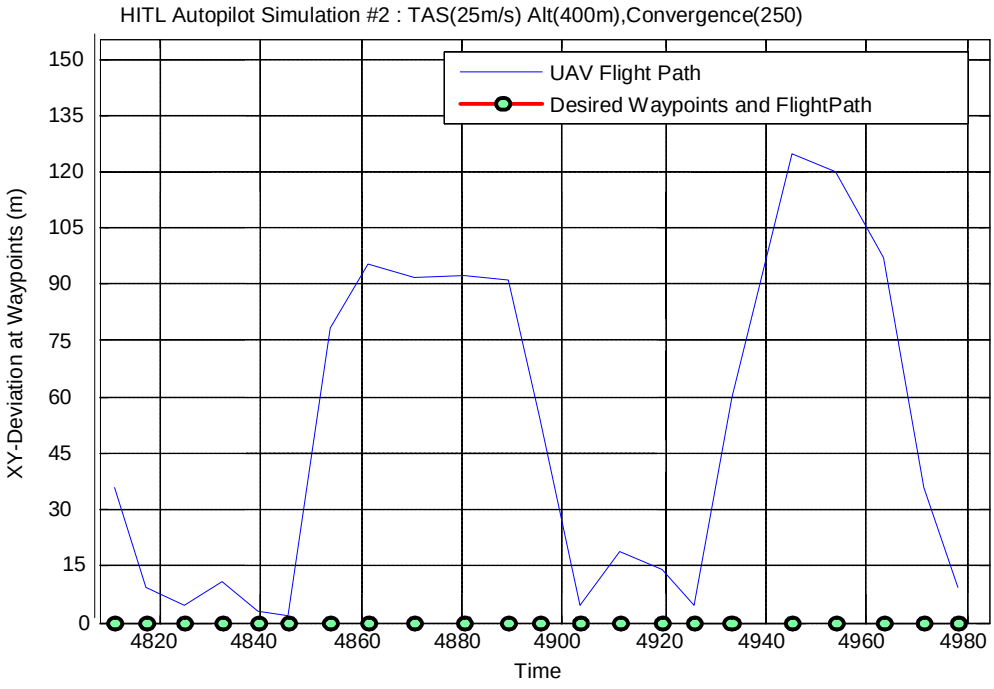

Figure 31 - Simulation #2 2-D Deviation at Waypoints

Obviously these flight paths are not satisfactory. The average 2-D deviation when the aircraft passes each waypoint is 49m, and reaches a maximum of 127m. The track convergence parameter was changed from 250 to 50 and the same track was flown again at the same airspeed. Figure 32 and Figure 33 show the results of this simulation.

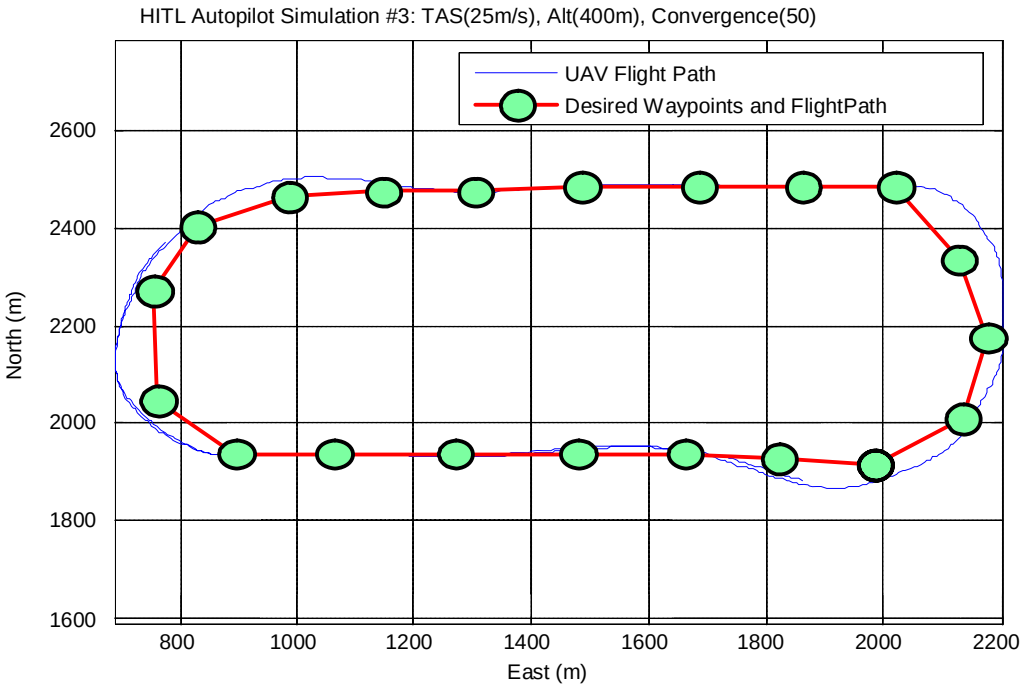

Figure 32 - Simulation #3 Flight Path

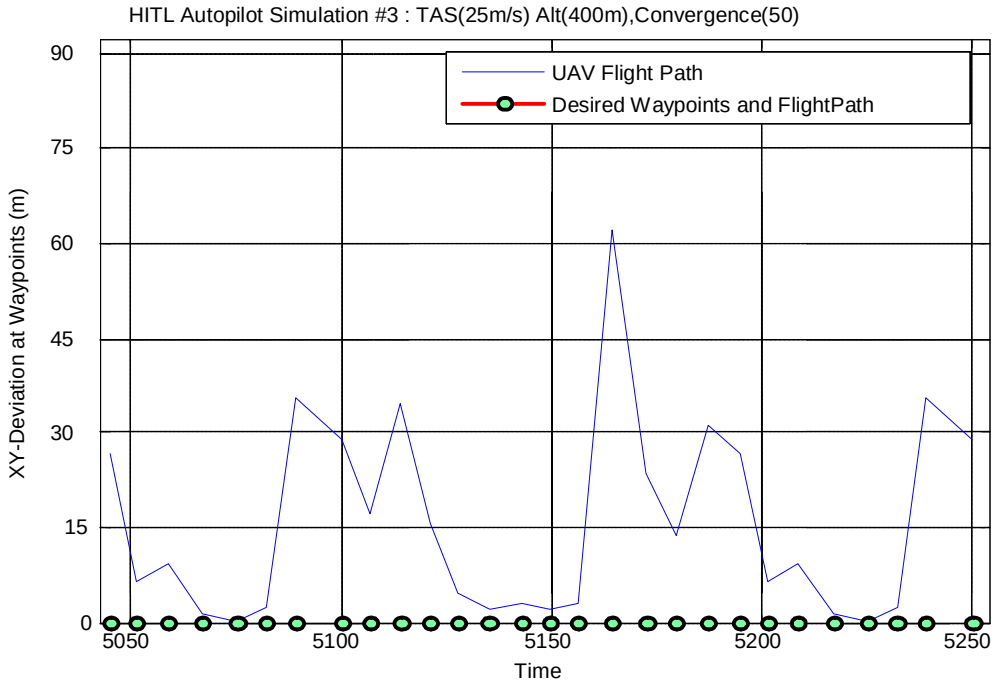

Figure 33 - Simulation #3 2-D Deviation at Waypoints

The results of this simulation are much better with a track convergence of 50, but are still not quite satisfactory. A maximum 2-D deviation at waypoints of 63m was achieved, and the average deviation was 16m. This sets the minimum radial separation of formation flight at 63m for airspeeds near 25m/s. This is the best performance that can hope to be achieved for such a high airspeed with the current gain settings.

Flying at such a high airspeed has proved through prior flight tests and simulations to yield unsatisfactory performance on this airframe; so most simulations will focus on lower airspeeds.

The performance capabilities at low airspeeds on a loose racetrack are also valuable to us, because they show the performance that might be achieved during mostly straight and level flight in a larger flight test area. Figure 34 and Figure 35 show the flight path and 2-D deviations at waypoints for an aircraft flying at 16m/s around a larger racetrack pattern with a track convergence of 50.

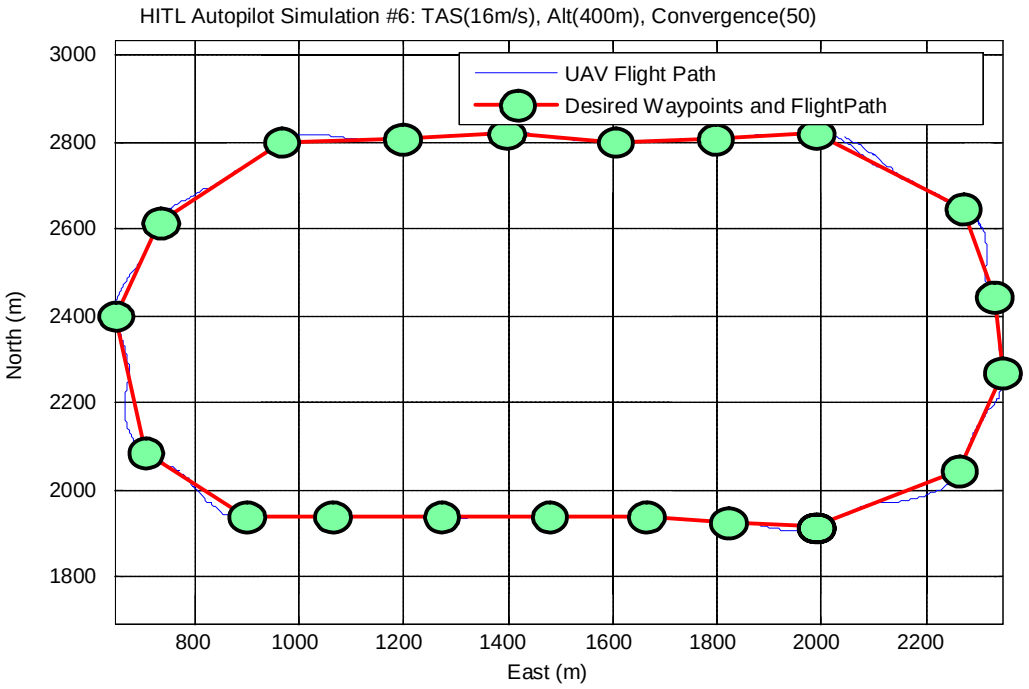

Figure 34 - Simulation #6 Flight Path

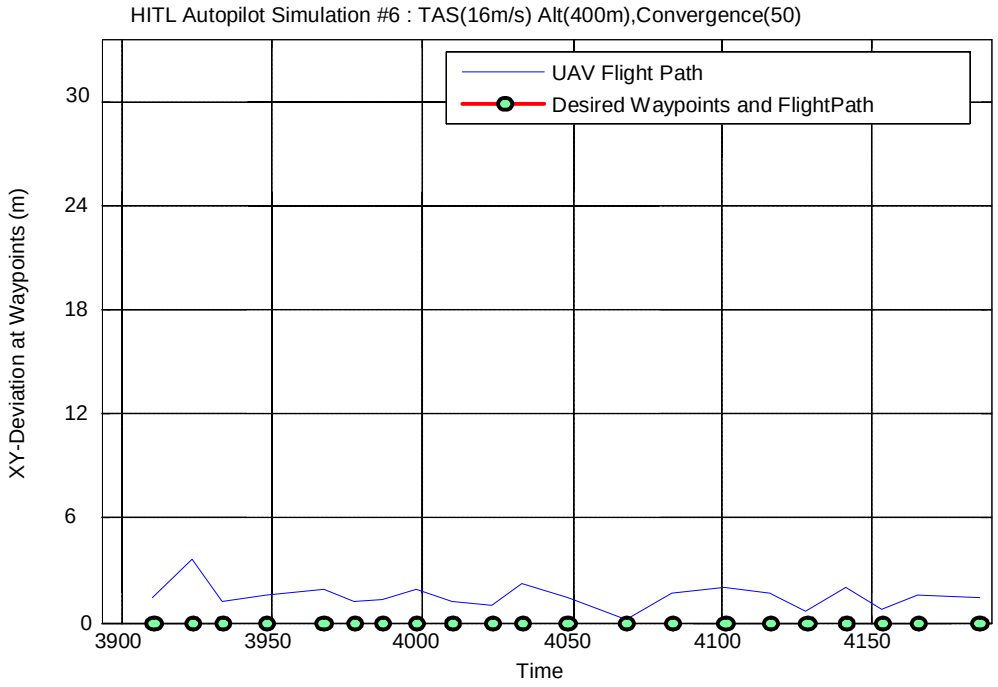

Figure 35 - Simulation #6 2-D Deviation at Waypoints

This simulation achieved excellent results under these conditions, with a maximum 2-D deviation of just 4m. Table 9 summarizes the performances achieved under the 6 different waypoint-following simulations.

Table 9 - Single Aircraft Waypoint-Following Simulation Results

Simulation # , Racetrack, Airspeed (m/s), Convergence	Average Altitude Deviation at Waypoints (m)	Maximum Altitude Deviation at Waypoints (m)	Average 2-D Deviation at Waypoints (m)	Maximum 2-D Deviation at Waypoints (m)
1, Tight, 16, 250	0.55	1.53	1.82	22.28
2, Tight, 25, 250	0.55	1.10	48.89	127.03
3, Tight, 25, 50	0.98	2.32	15.84	63.21
4, Loose, 25, 250	0.70	1.43	39.04	80.80
5, Loose, 16, 250	1.16	9.23	4.48	19.08
6, Loose, 16, 50	0.55	0.94	1.52	3.69

A number of trends can be seen in Table 9. First, simulations with a track convergence of 50 performed far better than those run with a convergence of 250. Additionally, tighter racetracks were flown better by simulations run at a slower airspeeds. The results of the turning radius simulations should verify the conclusion that slower airspeeds allow for better tracking. Altitude deviation does not seem to be a problem for any of the simulations. Simulation #5 had a higher maximum altitude deviation but it was an outlier, as shown in the relating graphs in Appendix A.

The turning radius simulations did in fact yield results that verified the conclusions drawn from the waypoint following simulations. Figure 36 and Figure 37

show the turning radius simulations flown at 16m/s and 25m/s, respectively. Simulations with varying track convergences were also run and the results of those can be found in Appendix A.

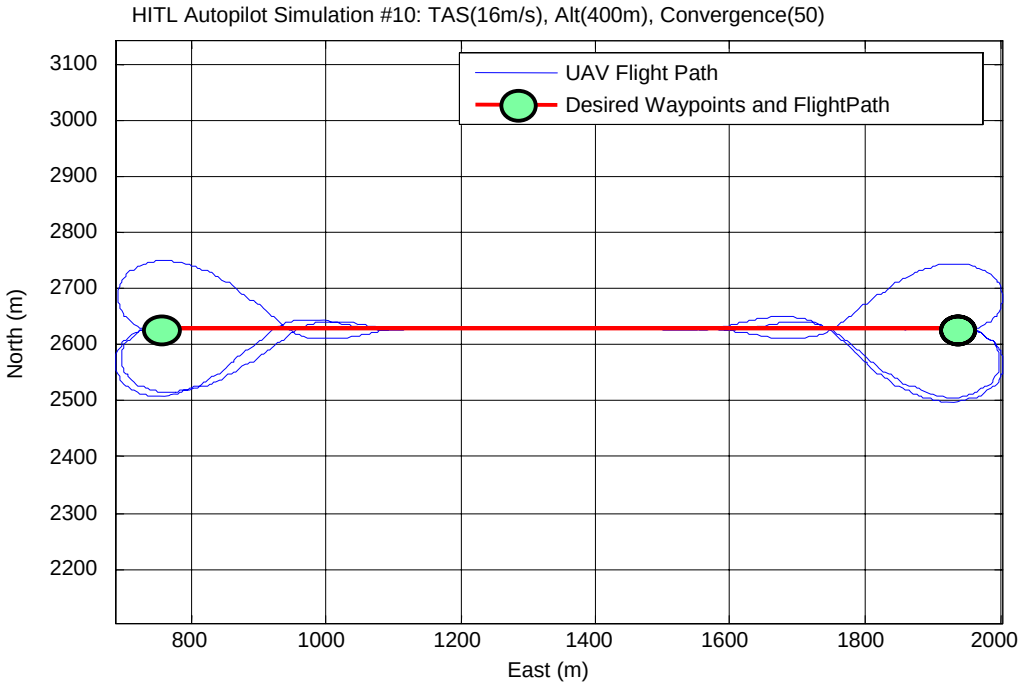

Figure 36 - Simulation #10 Turning Radius

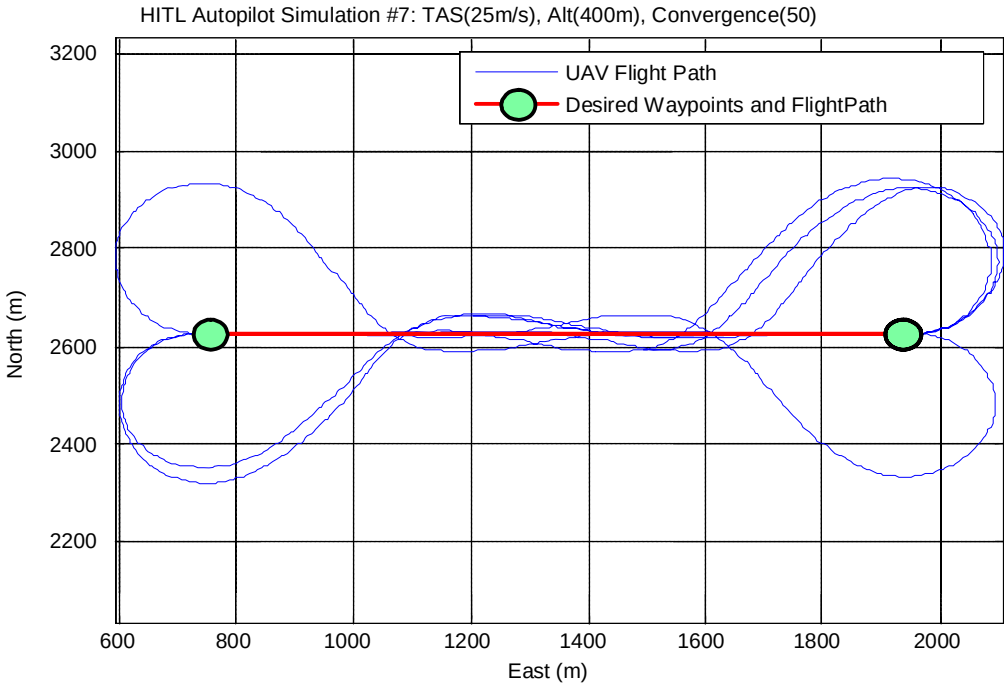

Figure 37 - Simulation #7 Turning Radius

The simulation flown at 16m/s yielded a minimum turning radius of approximately 110m, while the simulation flown at 25m/s had a turning radius of 300m. It must be noted that these radii were achieved while maintaining constant altitude. Much tighter turns could be accomplished if holding altitude was not a necessary condition. It was deemed necessary for this formation flight research, however, because the close formations that will be achieved do not allow for extreme maneuvers.

6.3 – Single Aircraft Flight Test Results

Time constraints and flight testing issues beyond the control of the author made flight testing impossible at this time. Flight testing will instead be accomplished by later research at AFIT. However, all simulation results should prove valid, as Jodeh's thesis shows (Jodeh, 2006).

6.4 – Formation Flight HITL Simulation Results

The results of the most noteworthy formation flight simulations are presented here; the others can be found in Appendix A.

Figure 38 and Figure 39 demonstrate probably the most important piece of information produced by this research – the time delay that exists when using a waypoint-guided autopilot for formation flight control.

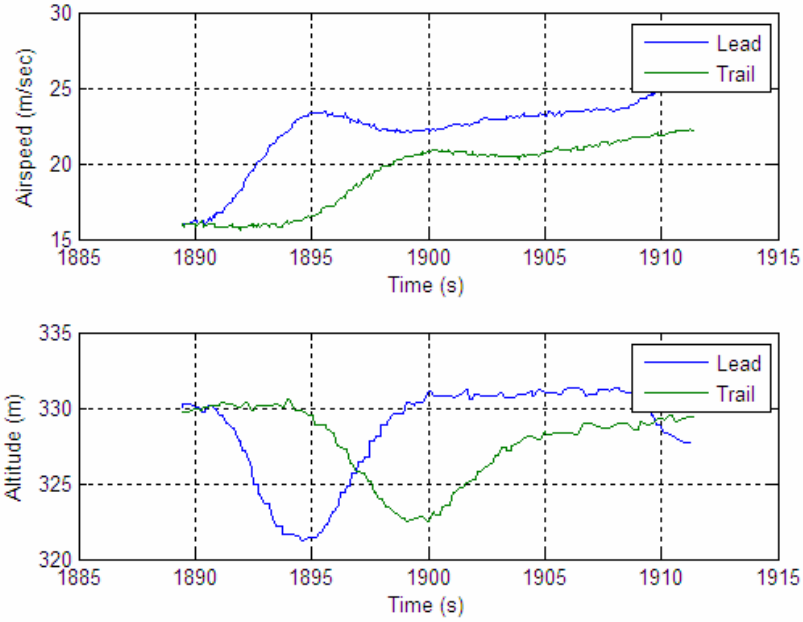

Figure 38 - Simulation #11 Time Delay: Airspeed Change

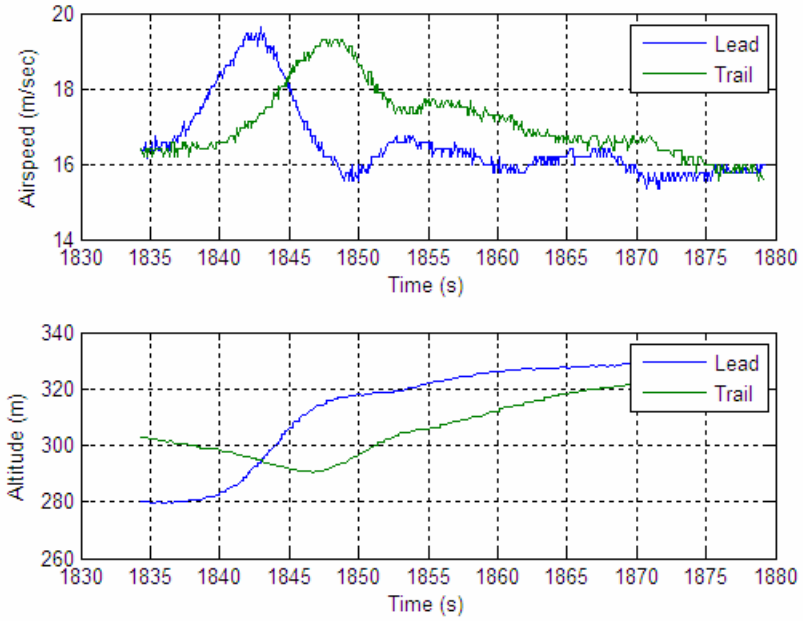

Figure 39 – Simulation #12 Time Delay: Altitude Change

Clearly there is a significant delay between the lead aircraft beginning its maneuver and the trail aircraft following accordingly. In both airspeed and altitude changes, the trail didn't begin its maneuver until between four and six seconds later. This will determine the precision of formations that can be achieved.

The next series of maneuvers consisted of varying airspeeds and track convergence parameters for the lead aircraft. Figure 40 and Figure 41 demonstrate that for a tight flight plan, the track convergence parameter of the lead aircraft has little effect on the formation capabilities. Because the aircraft is nearly at its minimum turning radius, the track convergence parameter does little to affect the path of the lead.

Another noteworthy performance issue with formation flight lies in the differing goals associated with formation flight and regular waypoint-following. The results of the single aircraft performance evaluation showed that lower track convergence parameters yielded significantly better performance in waypoint following, but this may actually have a negative effect on formation performance. Lower track convergence means the lead aircraft will be making more extreme maneuvers, making it harder for the trail aircraft to follow the lead. Therefore, a higher track convergence parameter of 250 should be used if formation flight is the primary design driver. If waypoint-following is more important, a lower track convergence parameter should be used.

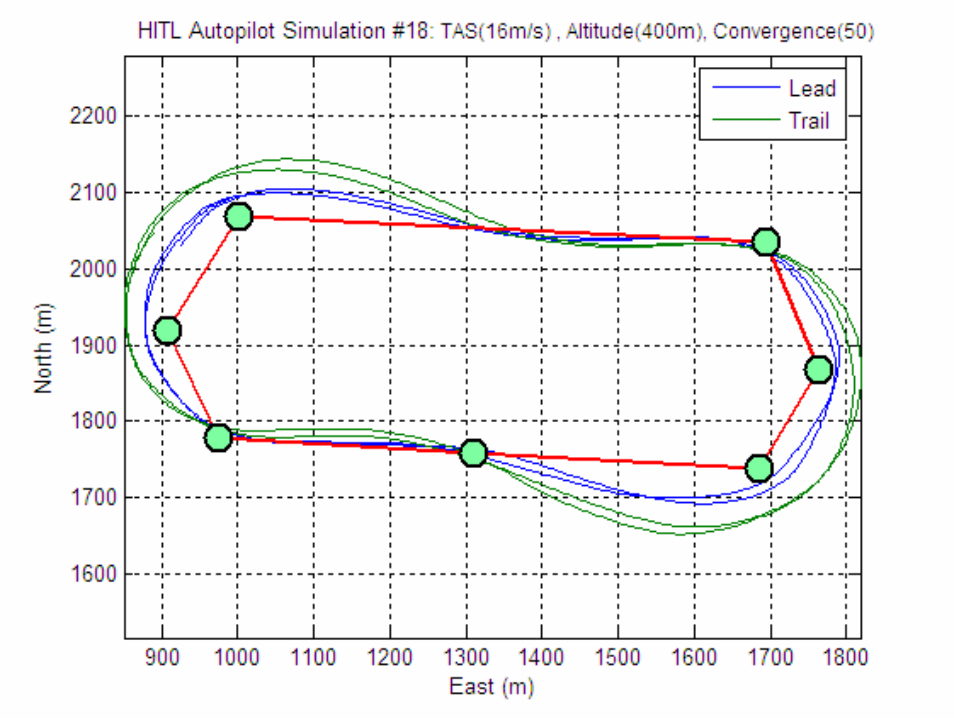

Figure 40 - Simulation #18 – Low Track Convergence during Formation Flight

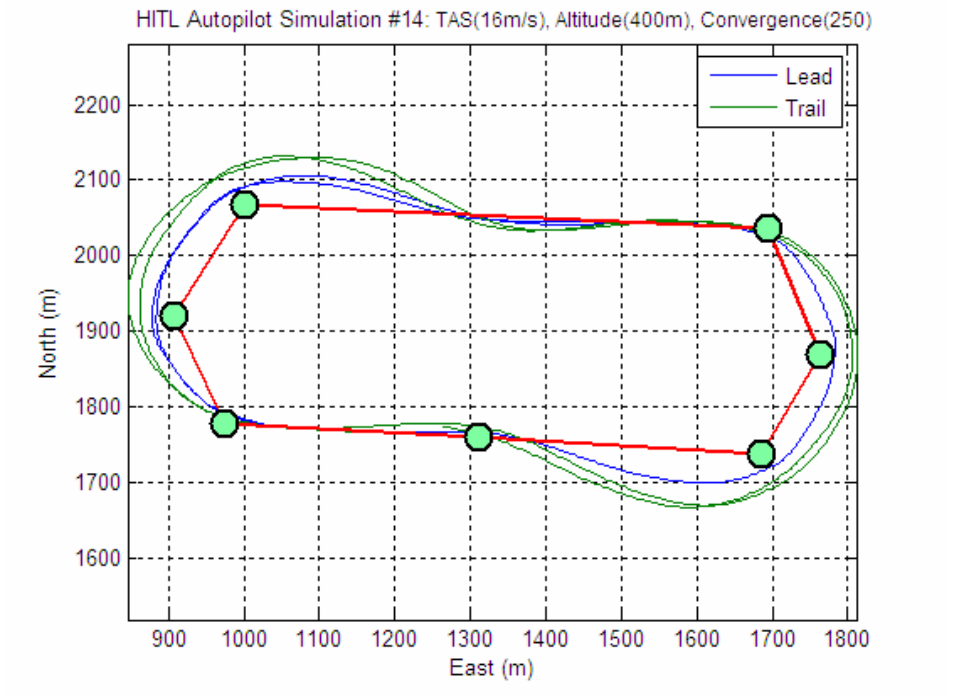

Figure 41 - Simulation #14 – High Track Convergence during Formation Flight

Figure 42 shows that higher airspeeds degrade formation performance. 25m/s is simply too great an airspeed to fly in the area designed by WPAFB.

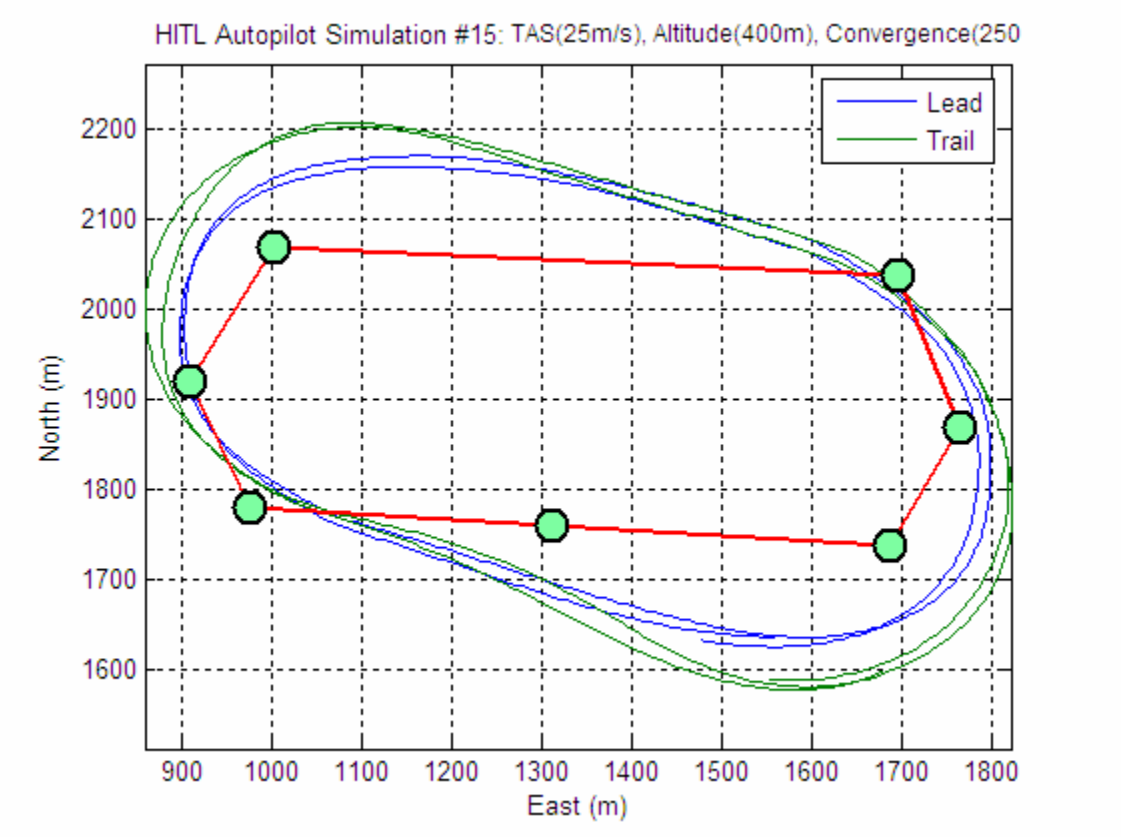

Figure 42 - Simulation #15 – Too High Airspeed

Several simulations were flown at a low airspeed of 12m/s, which proved to be too slow for formation flight using this particular aircraft. The aircraft stalls at airspeeds near 10m/s; so the autopilot is configured to not allow airspeeds below 12m/s. If a trail aircraft is following a lead too closely and receives a command to slow down, it may already be at its minimum airspeed, so there is a risk of collision. The formation flight algorithm corrects for this because it commands the trail aircraft to head to the waypoint that it has already passed, which makes the trail aircraft go into an immediate maximum

bank. When the trail gets far enough away so that the constantly updating waypoint is ahead of it rather than behind it, it retakes its position in the formation. Figure 43 shows that excellent formation flight was achieved during the majority of the flight, but that at two points the trail got too close. Those points are evidenced by the major deviations from the lead's flight path.

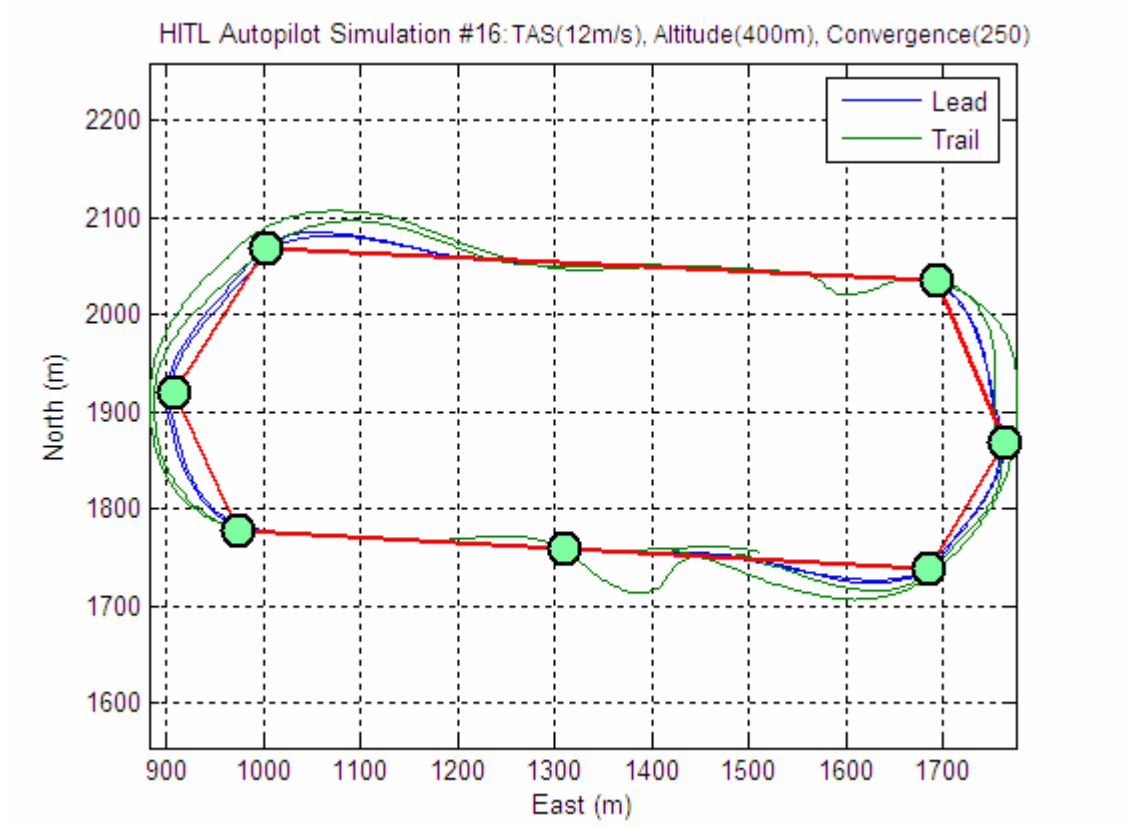

Figure 43 - Simulation #16 – Too Low Airspeed

The above simulations merely present a two-dimensional representation of each aircraft's flight path. Now the precise formations achieved will be presented.

Figure 44 and

Figure 45 shows the Area B flight plan being flown and the formations achieved using the stop-and-go method of airspeed adjustment. Figure 44 shows the separation between the trail and the desired waypoint, not necessarily the lead aircraft. However, because the “above” formation was studied during this simulation, the 2-D separation between the two aircraft and the 2-D separation between the trail aircraft and its desired waypoint are identical.

Figure 46 and Figure 47 show the same flight plan as Simulation #22 but with the logarithmic gain discussed in Section 3.4.2.

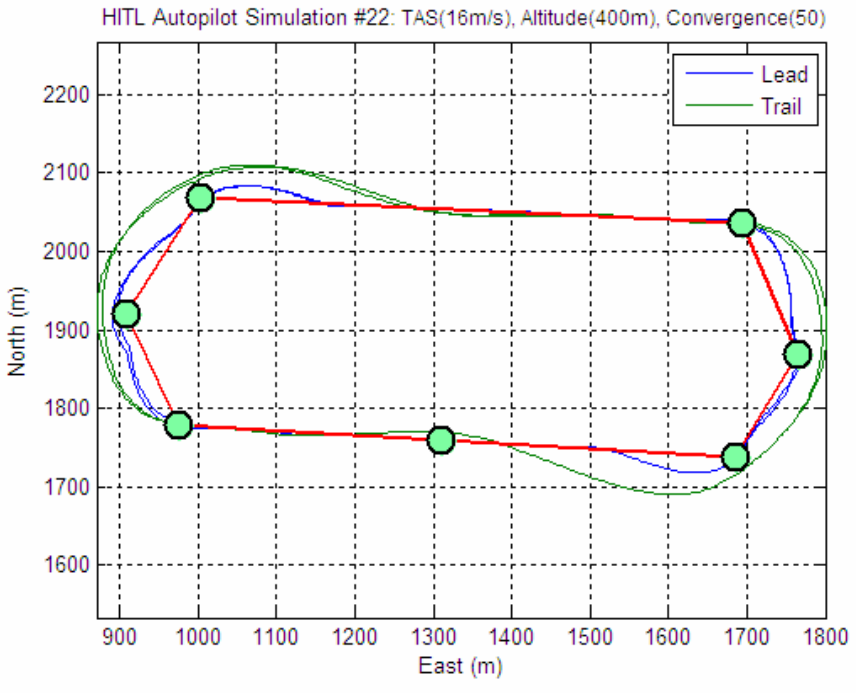

Figure 44 - Simulation #22 – Stop-and-Go Gain

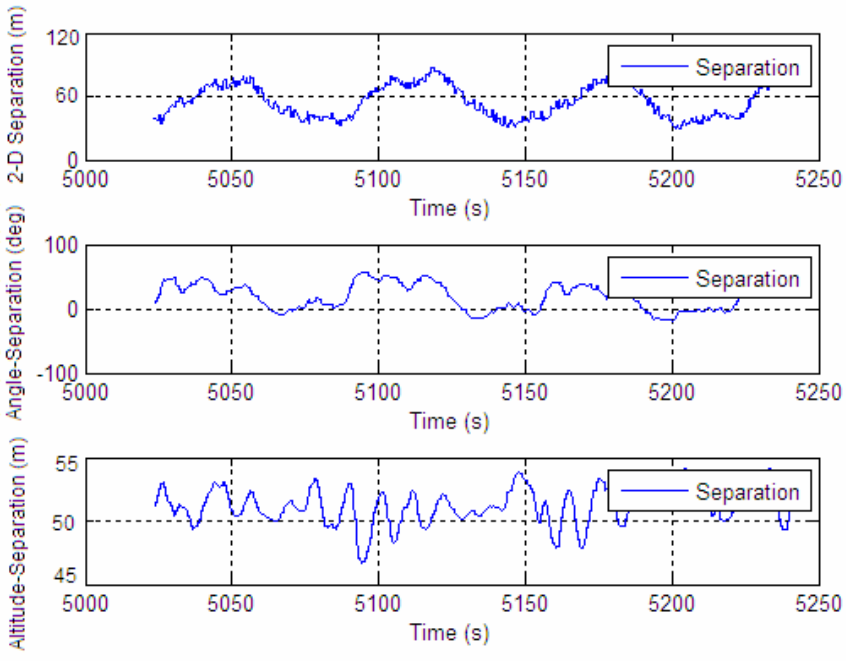

Figure 45 - Simulation #22 Formation Characteristics

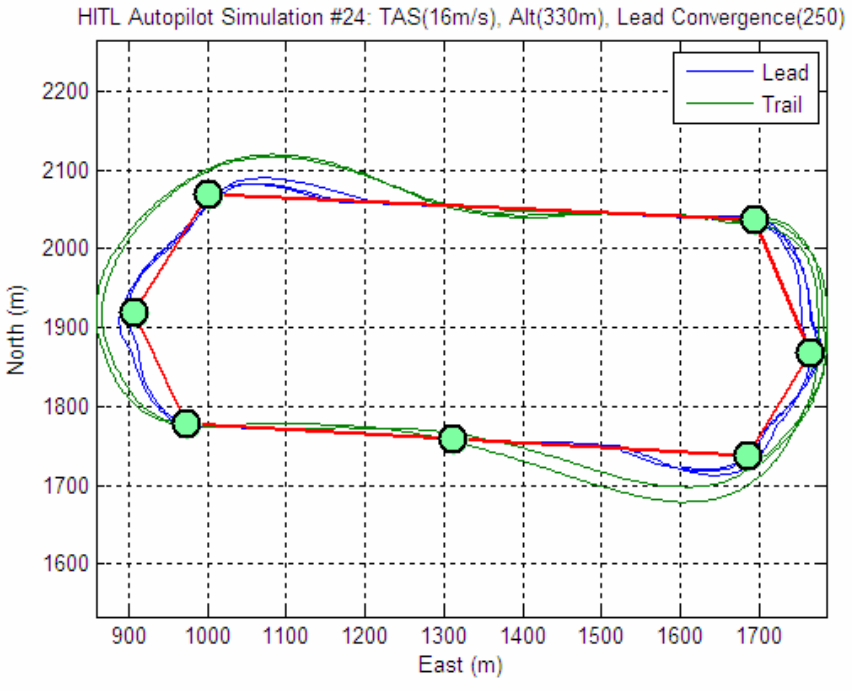

Figure 46 - Simulation #24 - Logarithmic Gain

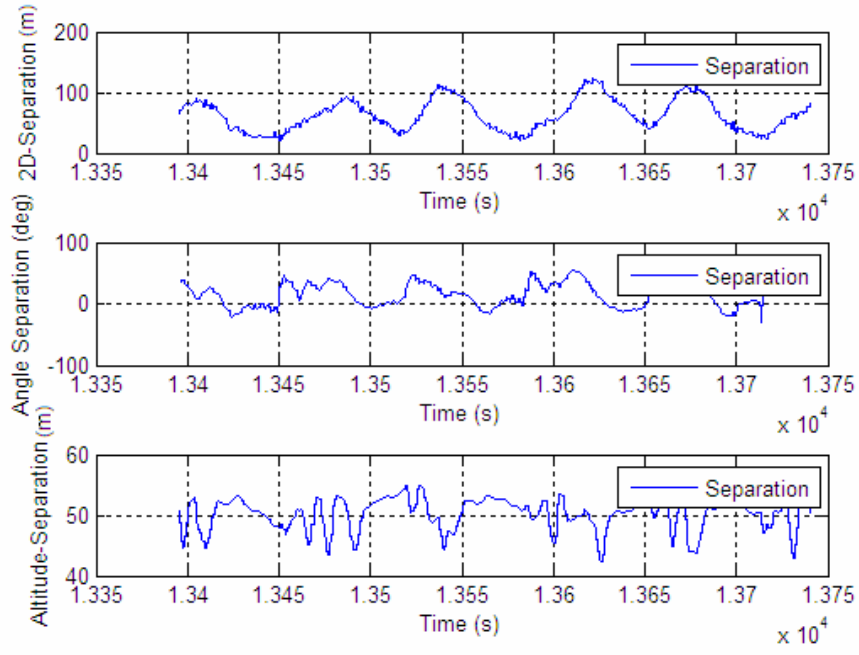

Figure 47 - Simulation #24 Formation Characteristics

Table 10 outlines the differences between using logarithmic gain and the supposedly rougher stop-and-go gain. The numbers correspond to Simulations #22 and #24, shown above. It must be noted that these simulations were run in the "Above" formation, where the trail aircraft's desired waypoint stays 50m directly above the lead.

	Logarithmic Gain	Stop-and-Go Gain
Average 2-D Separation (m)	63.04	56.65
Minimum 2-D Separation (m)	29.47	35.45
Maximum 2-D Separation (m)	122.49	89.14
Average Angle Separation (deg)	20.23	23.19
Maximum Angle Separation (deg)	55.78	57.38
Average Altitude Separation (m)	50.26	50.17
Maximum Altitude Separation (m)	57.44	53.88

Table 10 – Stop-and-Go versus Logarithmic Gain Adjustment

It is interesting to see that the two methods yielded virtually identical formation performances. This was bound to occur, as the logarithmic gain method was determined through a regression analysis of the stop-and-go gain method.

One thing that can be taken from Table 10 is that both methods have the same faults. The average angle separations are particularly troublesome because they are so high. This is due to the small flight testing area. A larger test area would perform

significantly better, as Figure 48 shows the results of simulation #19. Figure 49 shows the flight plan used in simulation #19.

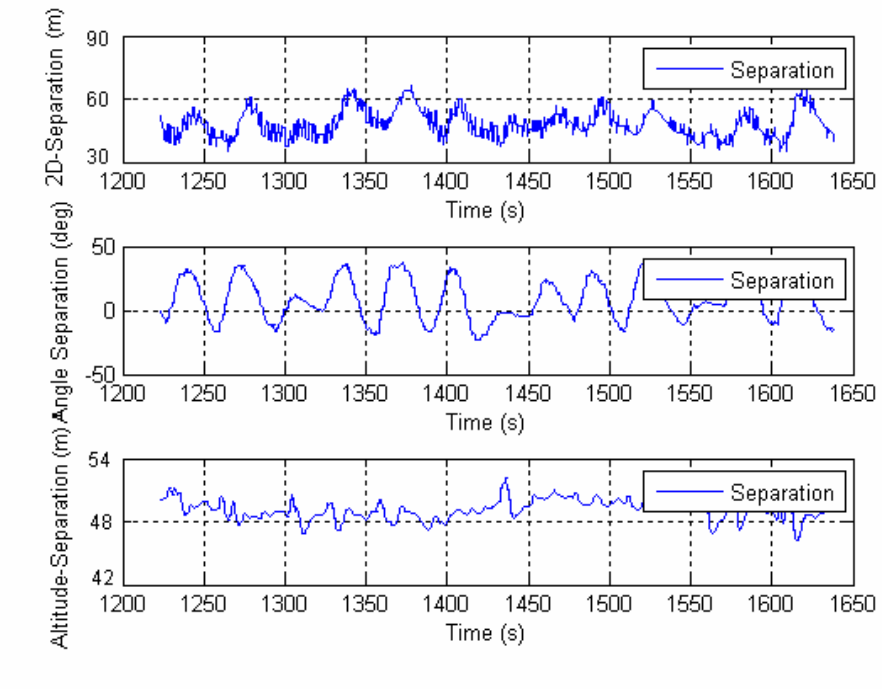

Figure 48 - Simulation #19 Formation Characteristics

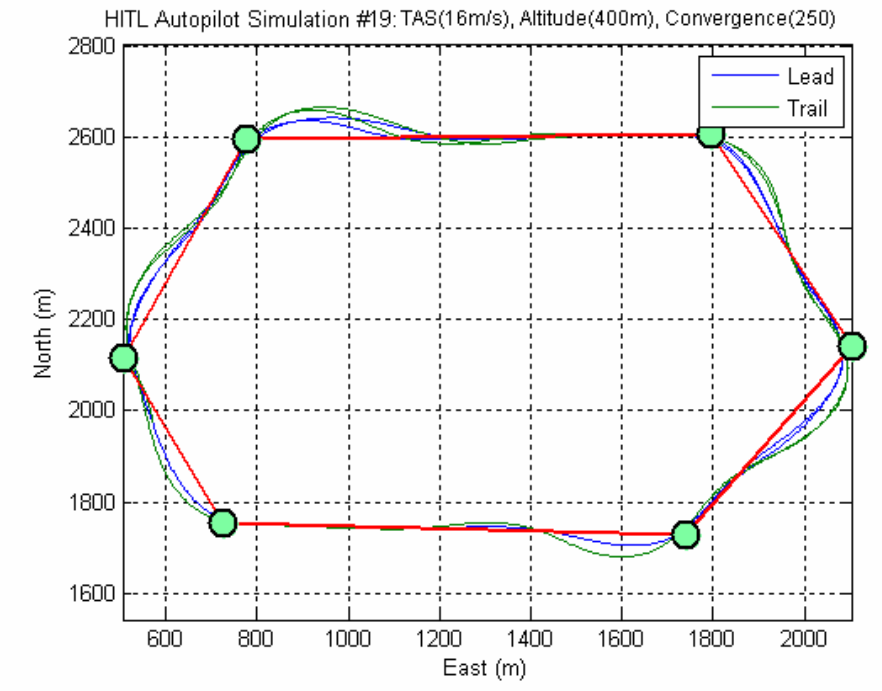

Figure 49 - Simulation #19 – Large Racetrack

This larger test area was flown with relative ease, flying with identical altitude and 2-D separation numbers as the previous simulations, but with an average bearing of just 13º and a maximum angle separation of just 40º. This shows the performance that could be achieved if the aircraft was restricted from making tight turns.

The important results from these formation flight simulations will now be summarized. The small flight testing area led to inferior performance at higher (25m/s) airspeeds. Therefore, formation flight should not be attempted at airspeeds higher than 20m/s. A higher track convergence parameter for the lead aircraft should be used when precision formation is desired, and a lower track convergence parameter should be used when waypoint-tracking is the primary goal. The time delay simulations show that there will be a four to six second lag between the lead and trail aircraft maneuvers. Therefore, formation precision will be limited to a separation of four to six times the airspeed. For these simulations, that limitation requires a separation of 64-96m based on an airspeed of 16m/s. The results outlined in Table 10 show separations of 57 and 63m for each of the two methods, which confirms the time delay. This time delay does not necessarily mean that tighter formations are not possible; it simply means that the maneuvers of the lead aircraft will have to take into account the delayed maneuvers of the trail aircraft. For instance, Figure 50 depicts a lead aircraft in a right turn while a trail aircraft maintains too close of a separation in a 'V' formation. Similar situations could occur for each of the predefined formations.

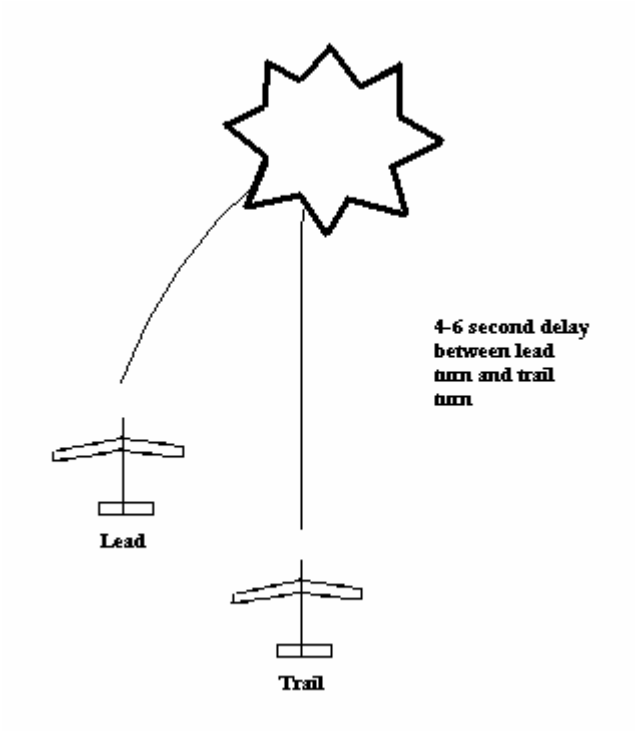

Figure 50 - Time Delay Collision

The problem depicted above is slightly alleviated due to the way the formation flight algorithm was designed. The trail aircraft's waypoint will initially swing the opposite direction to the lead aircraft's turn because its waypoint is based off of the lead aircraft's reference frame. So in the event of a lead aircraft in a right turn, the following things will happen:

- Lead aircraft begins a right turn.
- Formation separation becomes too small and bearing becomes too large, which commands a lower airspeed to the trail.
- Trail desired waypoint swings slightly left while lead aircraft goes ahead and passes in front of trail.

- Trail aircraft gets back within proper bearing range and speeds up, retaking ‘V’ position alongside the lead

This is the process that should occur if proper formation separations are defined. With the proper formation separations and tuned gains, there is little chance of collision.

Once the time delay issue has been addressed and much more precise formation flight is possible, the “swinging” of a waypoint during a lead aircraft maneuver becomes more of a problem than a solution. A recent thesis was published at AFIT that looked into this problem as it related to automated aerial refueling (Ross, 2006).

6.5 – Formation Flight Test Results

As stated before, experimental flight testing was not possible at this time. However, simulation has verified that the formation flight algorithm is functional, and there is no reason why the results should not translate over to real-world flight testing. The important things to take from the simulations are the time delays involved in UAS operation and their relationship with the formation flight algorithm.

6.6 – Chapter Conclusions

The simulated single aircraft performance evaluation showed the limits that could be achieved during formation flight. Those limits were applied to the formation flight algorithm, and various conditions were studied through simulation. Unfortunately, the conclusions drawn from simulation could not be validated through experimental flight testing, but there is no reason to suggest that the results would be any different.

VII. Conclusions and Recommendations

7.1 – Chapter Overview

This chapter will describe the major accomplishments of this thesis, as well as make a number of suggestions for follow-on projects and recommendations for future work to improve the UAV test bed at AFIT.

7.2 – Conclusions

This research has accomplished several key objectives:

- Implementation of the SDK into autonomous UAV flight
- Establishment of procedures for characterizing guidance and control performance limits for varying conditions of a single UAV operating autonomously
- Achievement of waypoint-guided formation flight using established performance limits to tune gains
- Established procedures for further work using the SDK

The single aircraft performance tests established limits for varying flying conditions, and were incorporated into the formation flight algorithm. Unfortunately, flight testing was not possible during this thesis, so the simulations will be validated by the AFIT UAV research group at a later date.

7.3 – Recommendations

The following recommendations cover aspects of formation flight and interaction with the avionics through the SDK. Additionally, recommendations for future research projects are suggested.

- Use optimization to fine-tune Piccolo II autopilot gains. A guess-and-check method was used in a prior thesis and achieved satisfactory results, but since extensive flight testing has been done since then, optimization could tune gains based on the existing data.
- Develop a user interface that is more capable of controlling the avionics. The Win32 Application is not nearly as user-friendly as a GUI should be.
- Formation Flight Recommendations
    - Optimize commands to UAS instead of only waypoint commands followed by airspeed and altitude adjustments.
    - Connect airborne avionics through wireless networks in onboard computers so communication through the ground station is unnecessary. The ground station communicates at only 1Hz, which is not fast enough to send constantly updating commands to achieve smooth, close formation flight.
    - Investigate further the effect of track convergence parameter on formation flight.
- Flight Test Recommendations
    - Larger flight test area would be ideal, especially for formation flight.

Appendix A – Flight Test Results

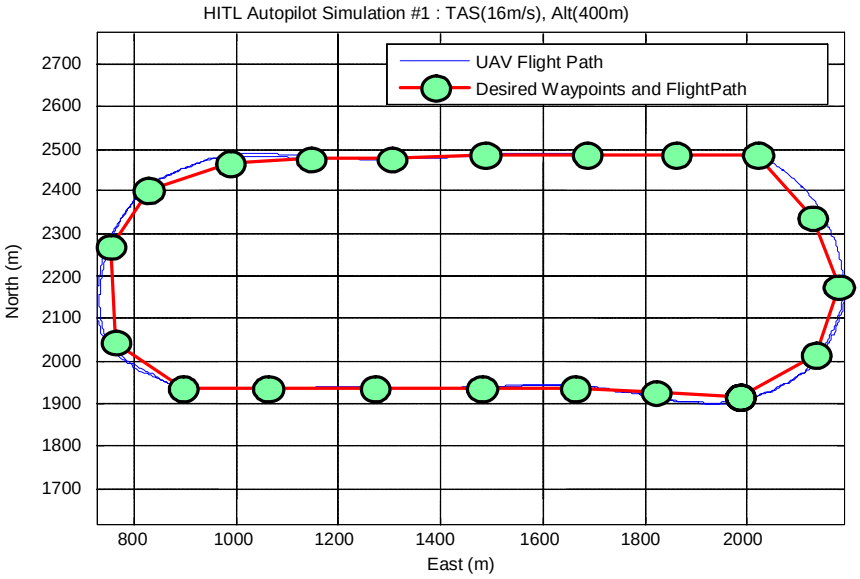

Figure 51 - Simulation #1 – Airspeed 16m/s, Track Convergence 250

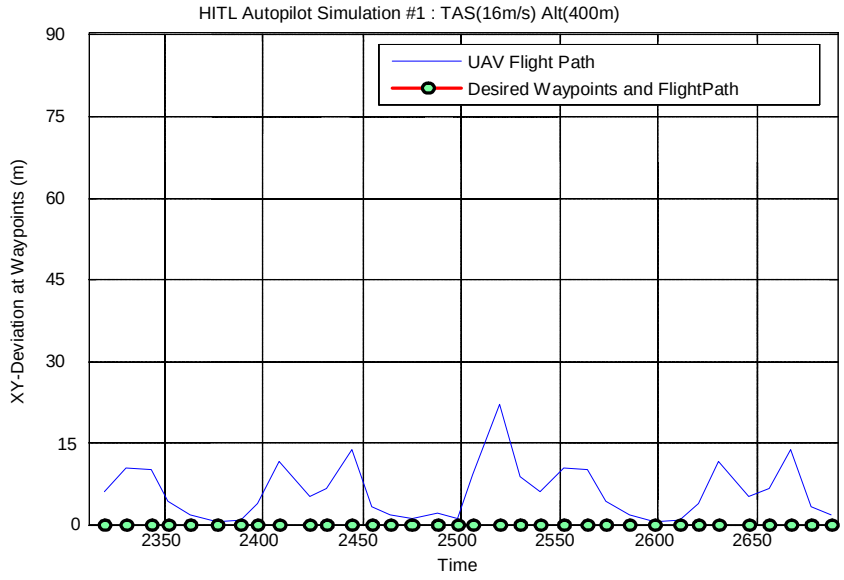

Figure 52 - Simulation #1 Airspeed 16m/s, Track Convergence 250

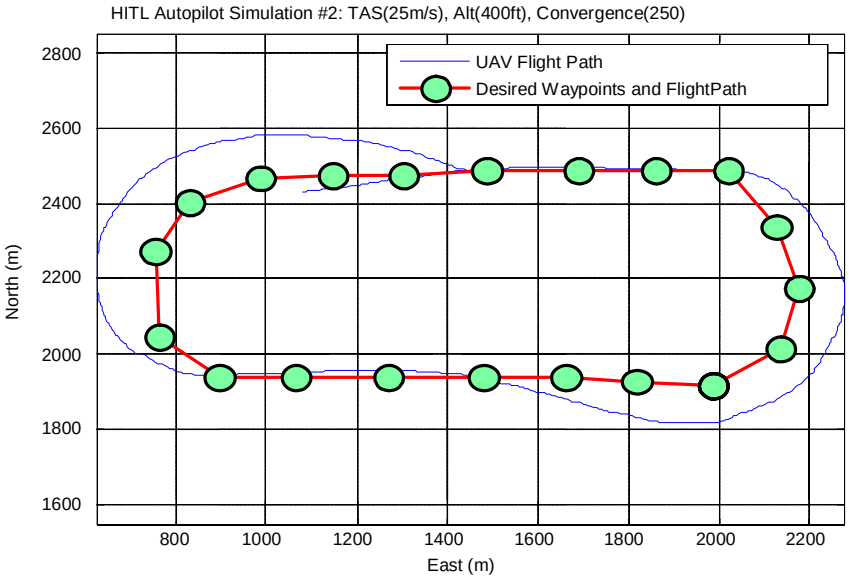

Figure 53 - Simulation #2 Airspeed 25m/s, Track Convergence 250

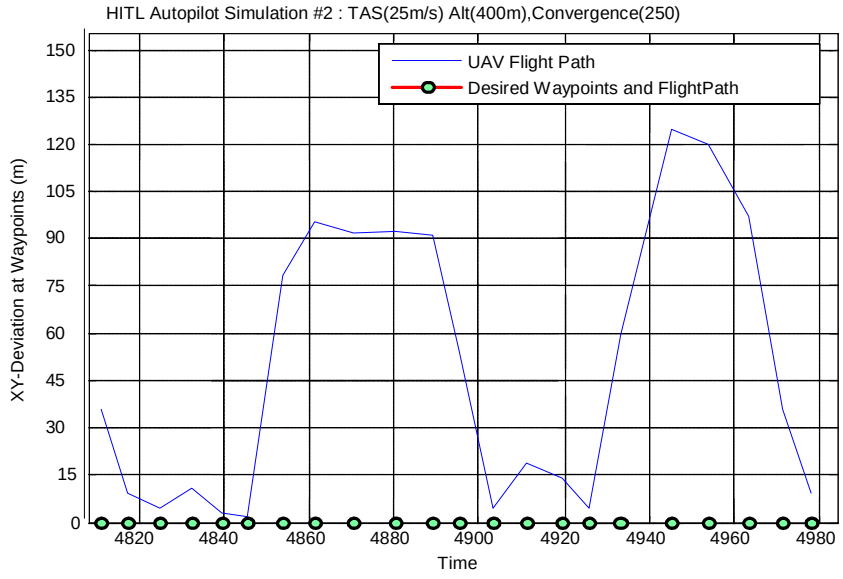

Figure 54 - Simulation #2 Airspeed 25m/s, Track Convergence 250

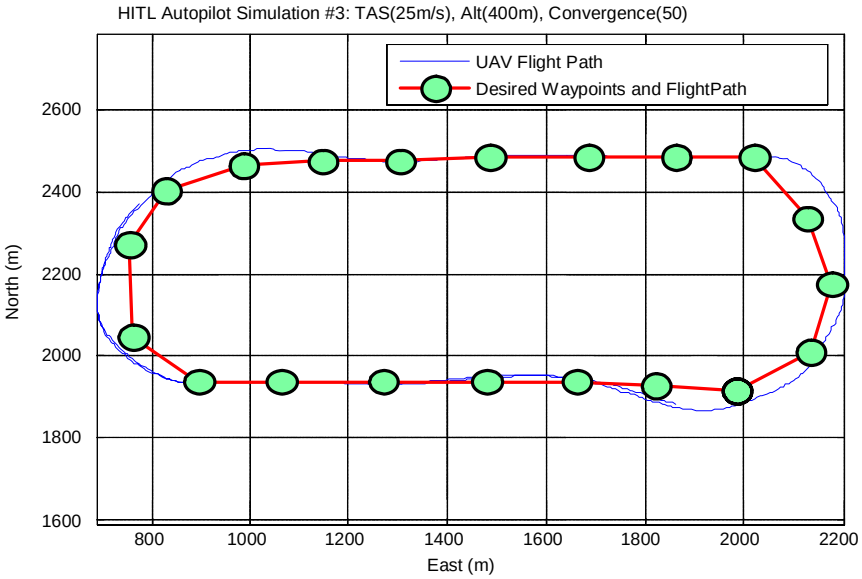

Figure 55 - Simulation #3 Airspeed 25m/s, Track Convergence 50

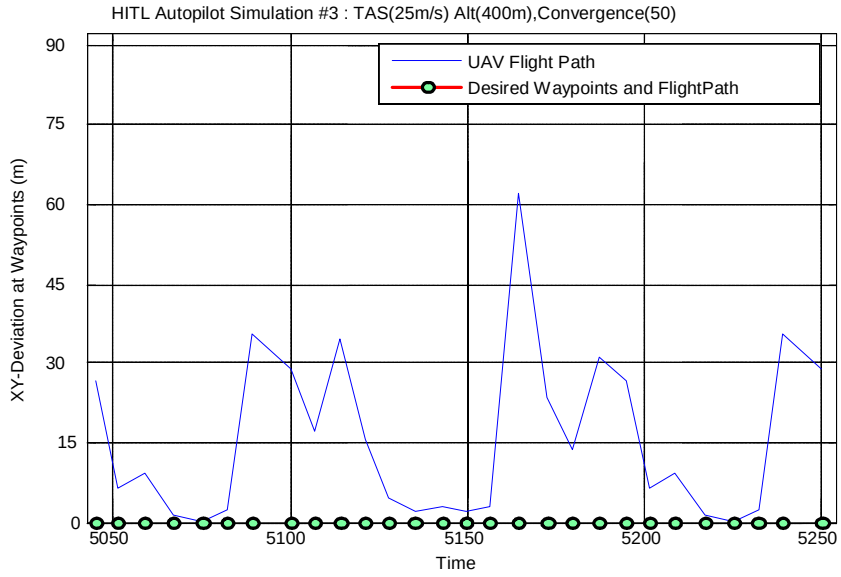

Figure 56 - Simulation #3 Airspeed 25m/s, Track Convergence 50

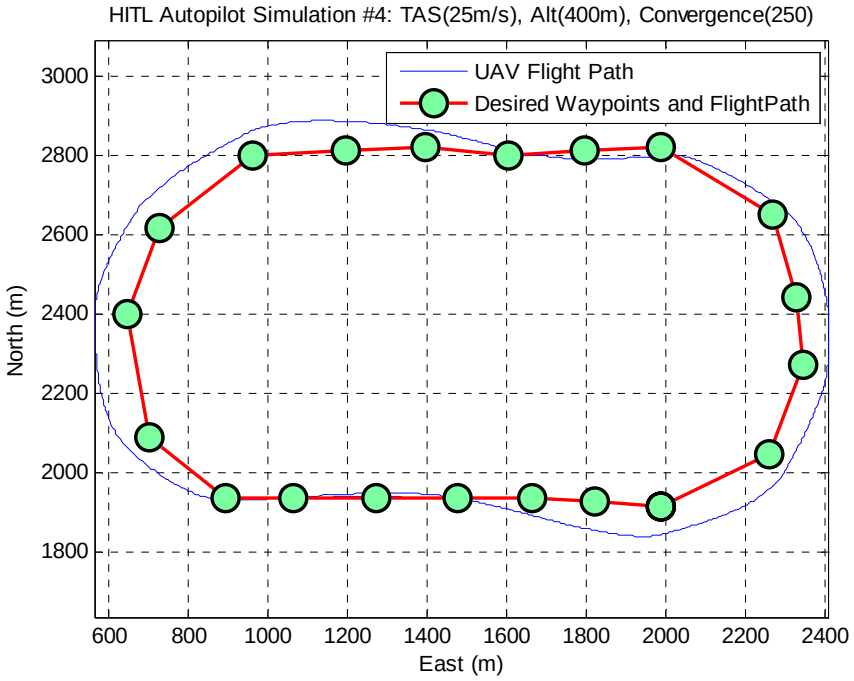

Figure 57 - Simulation #4 Airspeed 25m/s, Track Convergence 250

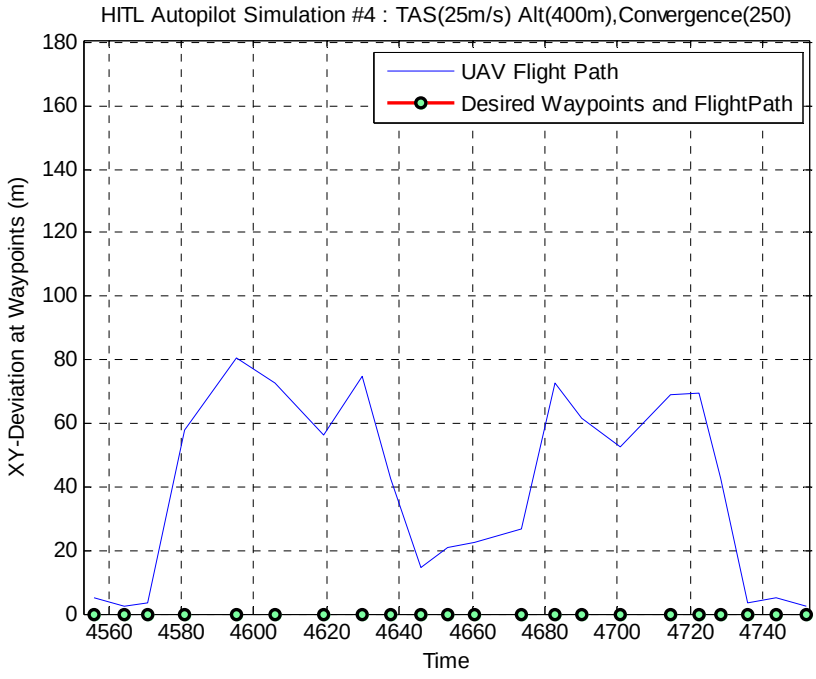

Figure 58 - Simulation #4 Airspeed 25m/s, Track Convergence 250

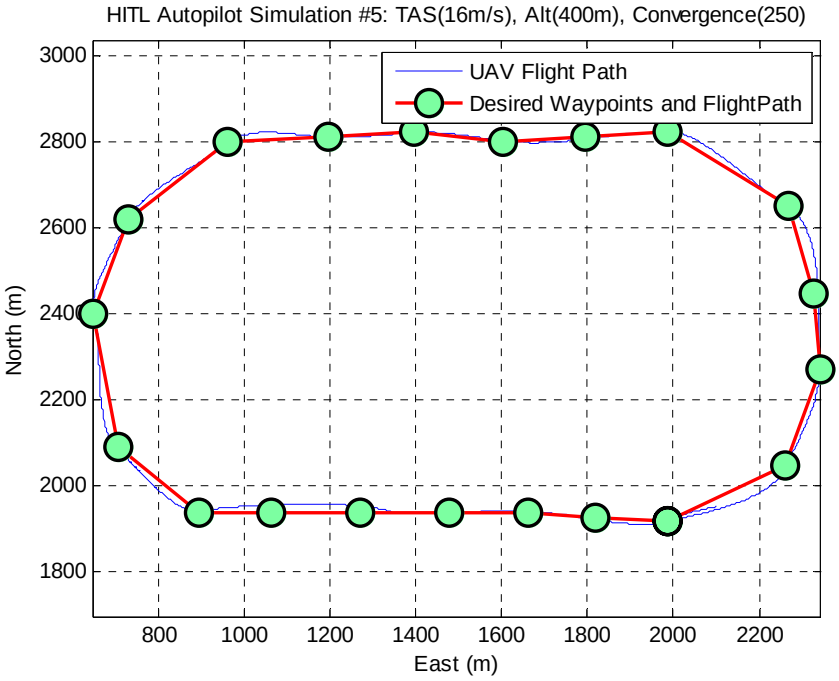

Figure 59 - Simulation #5 Airspeed 16m/s, Track Convergence 250

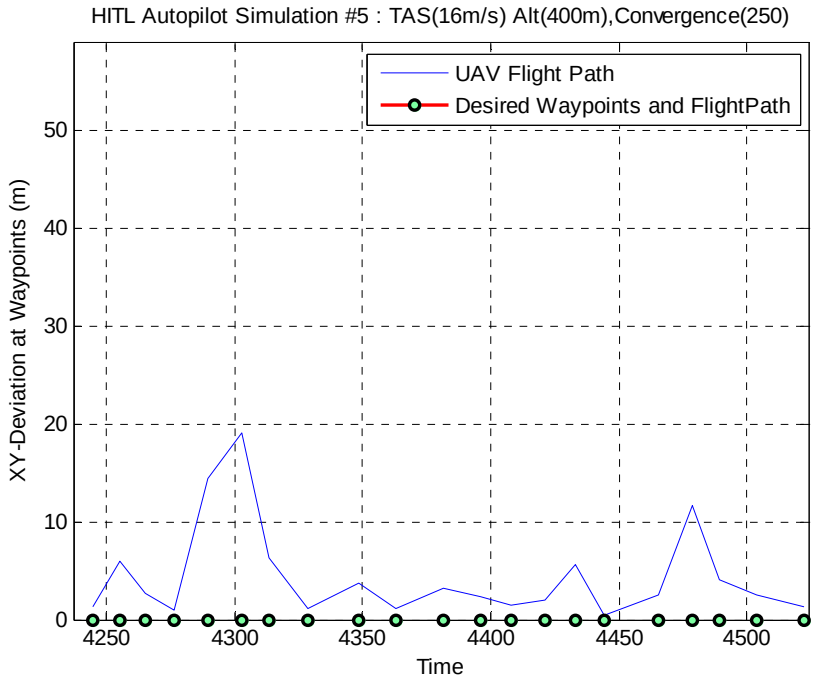

Figure 60 - Simulation #5 Airspeed 16m/s, Track Convergence 250

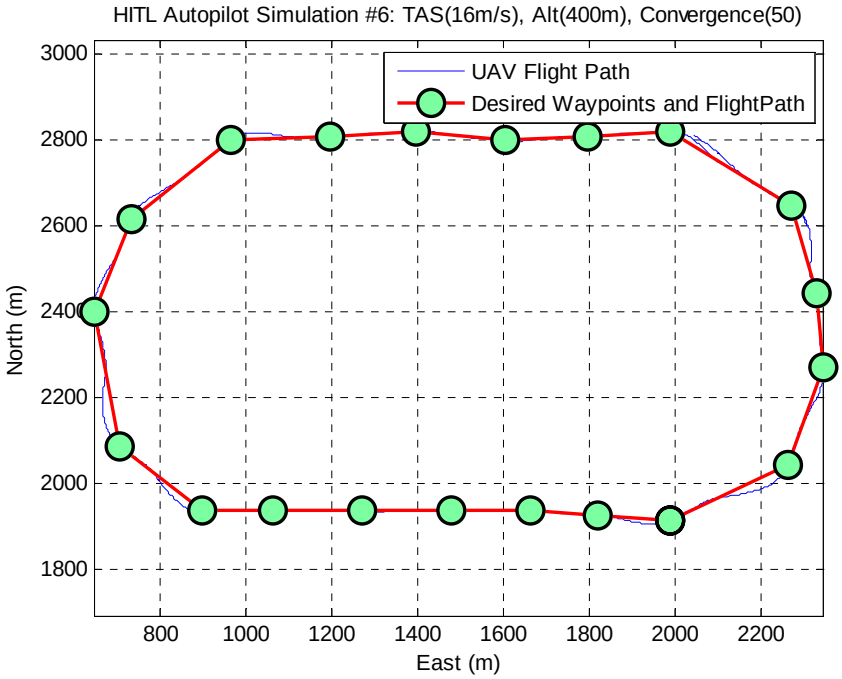

Figure 61 - Simulation #6 Airspeed 16m/s, Track Convergence 50

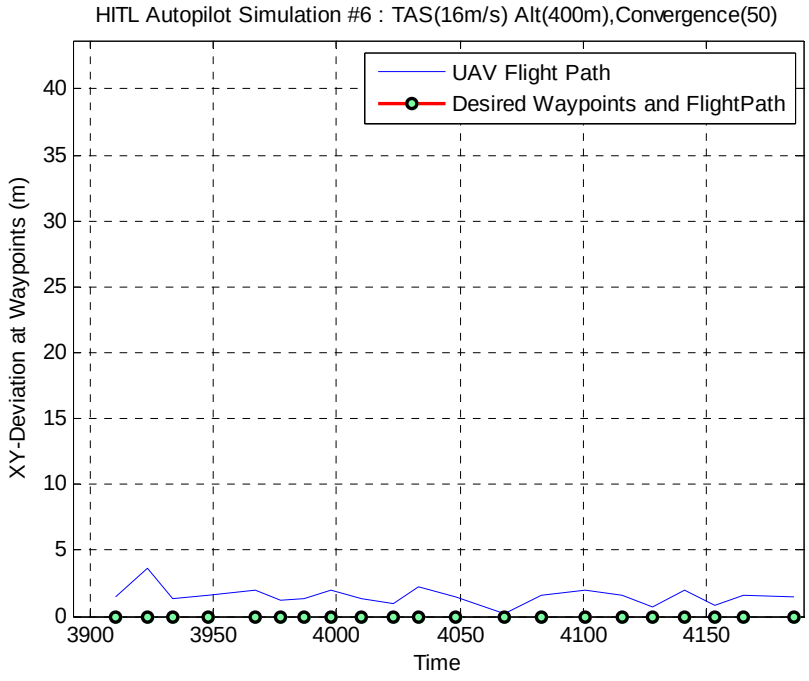

Figure 62 - Simulation #6 Airspeed 16m/s, Track Convergence 50

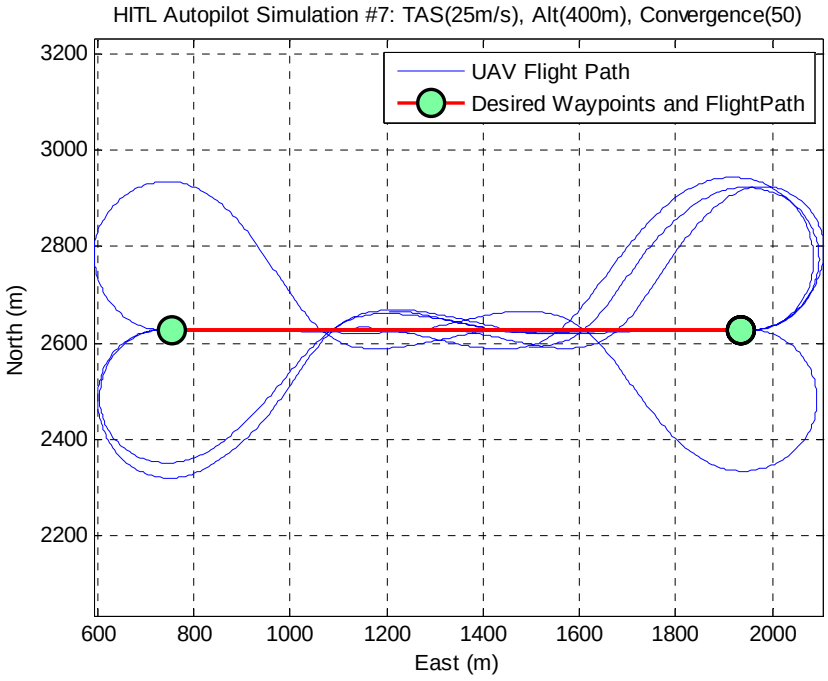

Figure 63 - Simulation #7 Airspeed 25m/s, Track Convergence 50

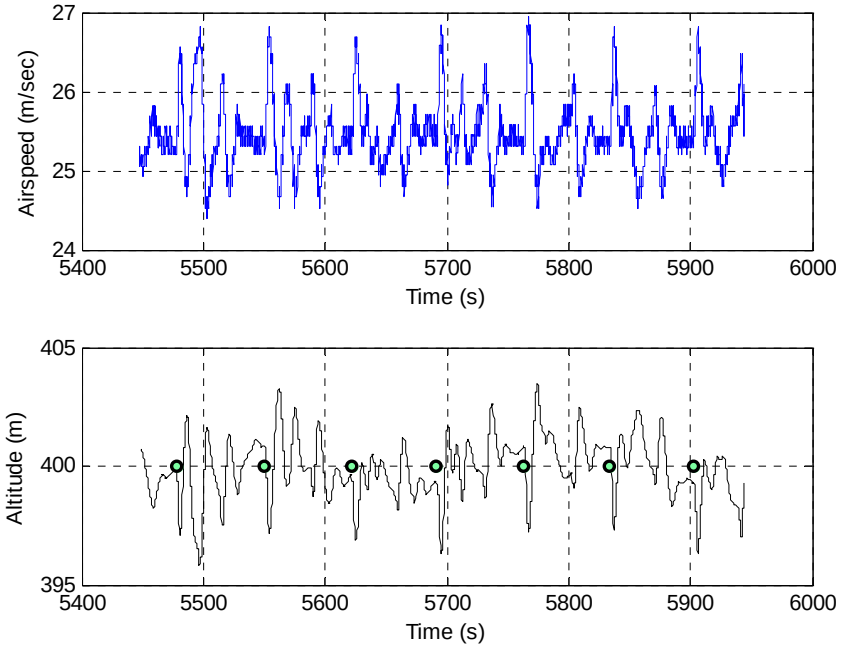

Figure 64 - Simulation #7 Airspeed 25m/s, Track Convergence 50

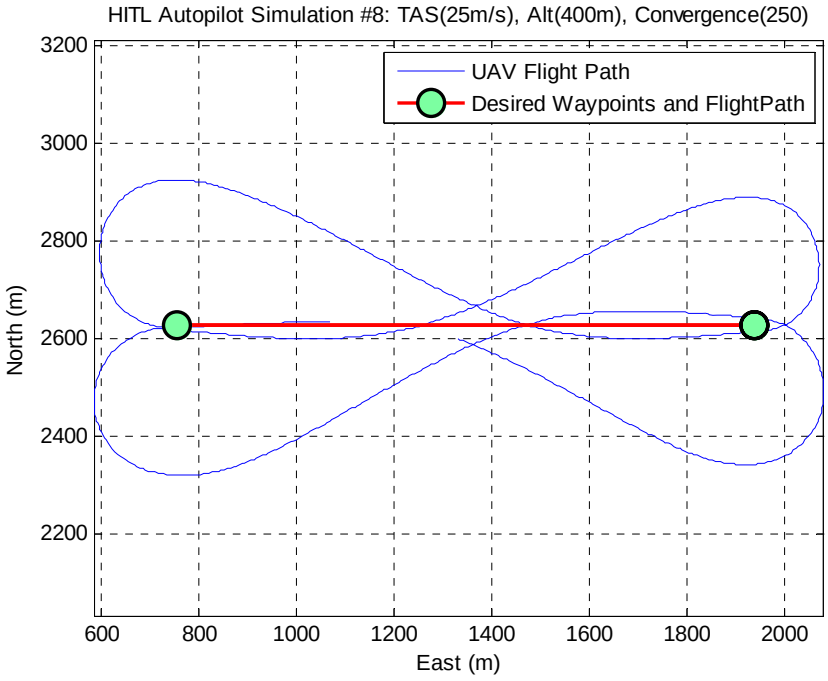

Figure 65 - Simulation #8 Airspeed 25m/s, Track Convergence 250

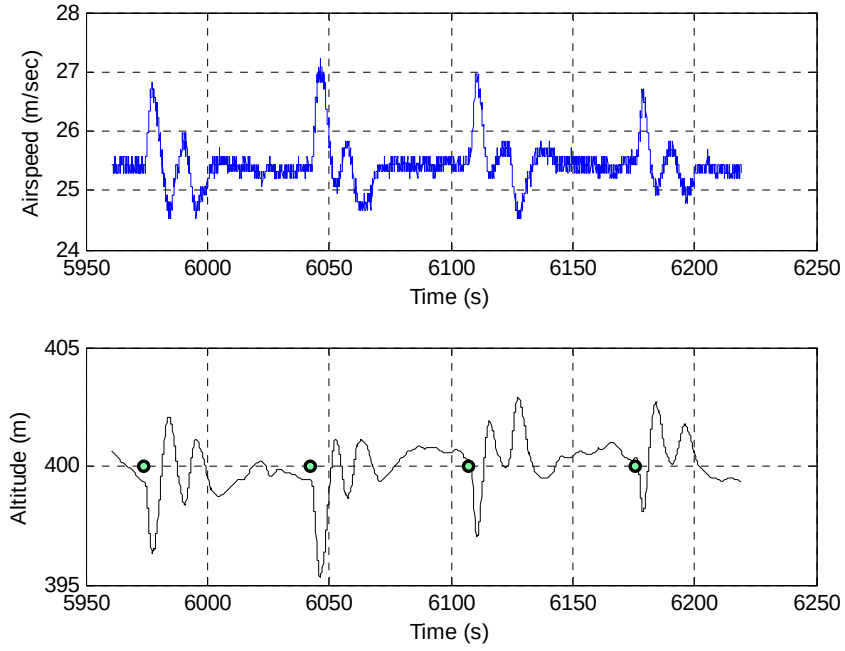

Figure 66 - Simulation #8 Airspeed 25m/s, Track Convergence 250

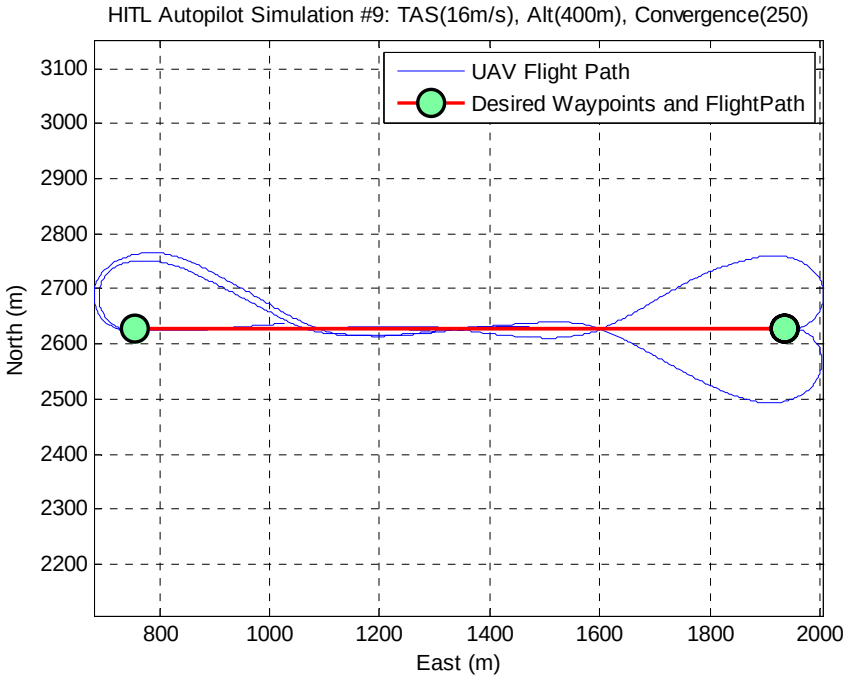

Figure 67 - Simulation #9 Airspeed 16m/s, Track Convergence 250

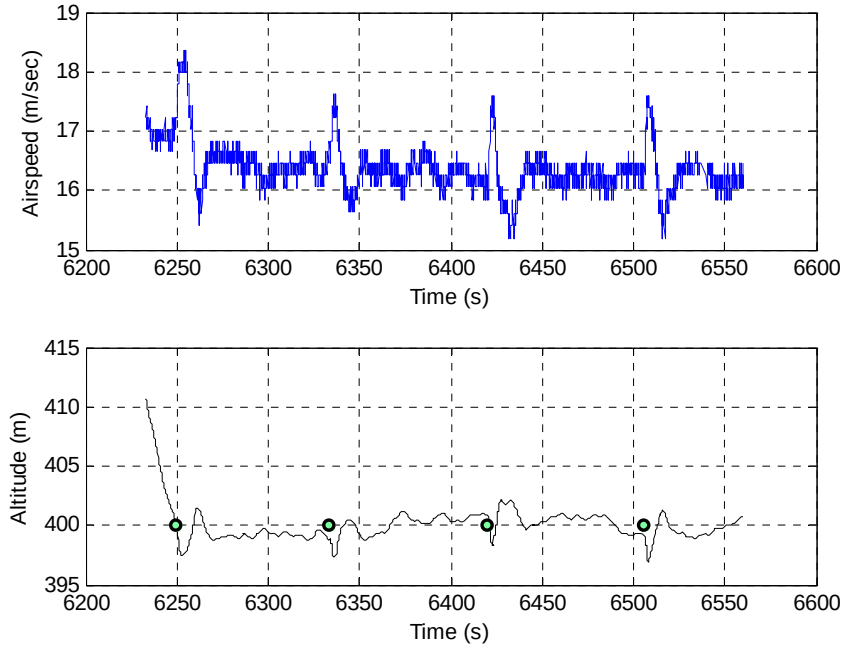

Figure 68 - Simulation #9 Airspeed 16m/s, Track Convergence 250

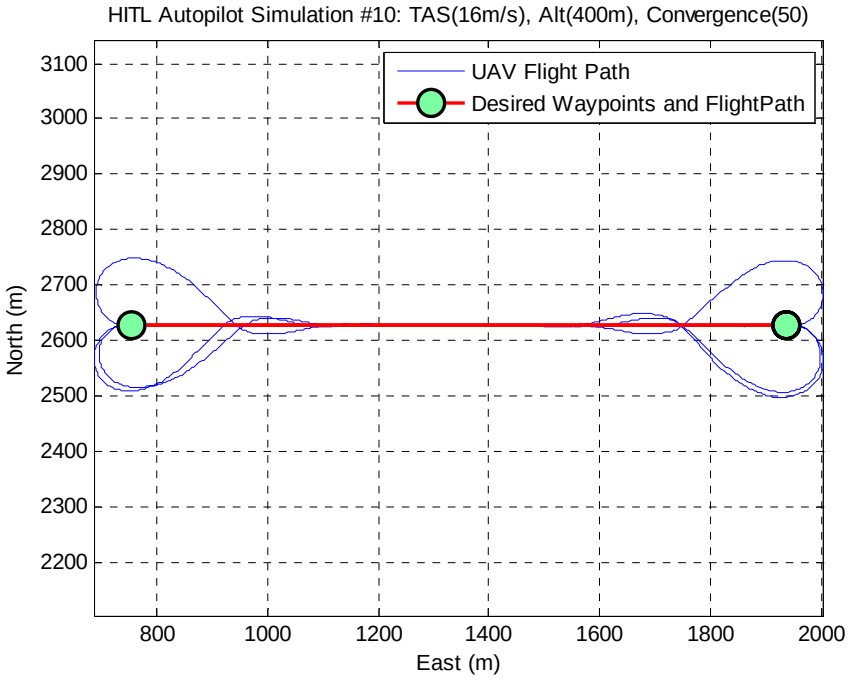

Figure 69 - Simulation #10 Airspeed 16m/s, Track Convergence 50

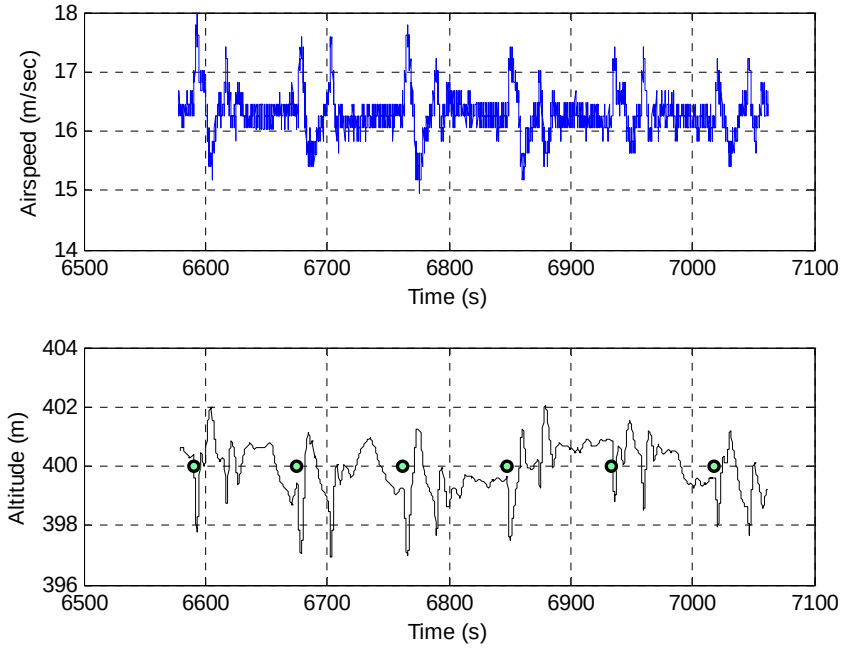

Figure 70 - Simulation #10 Airspeed 16m/s, Track Convergence 50

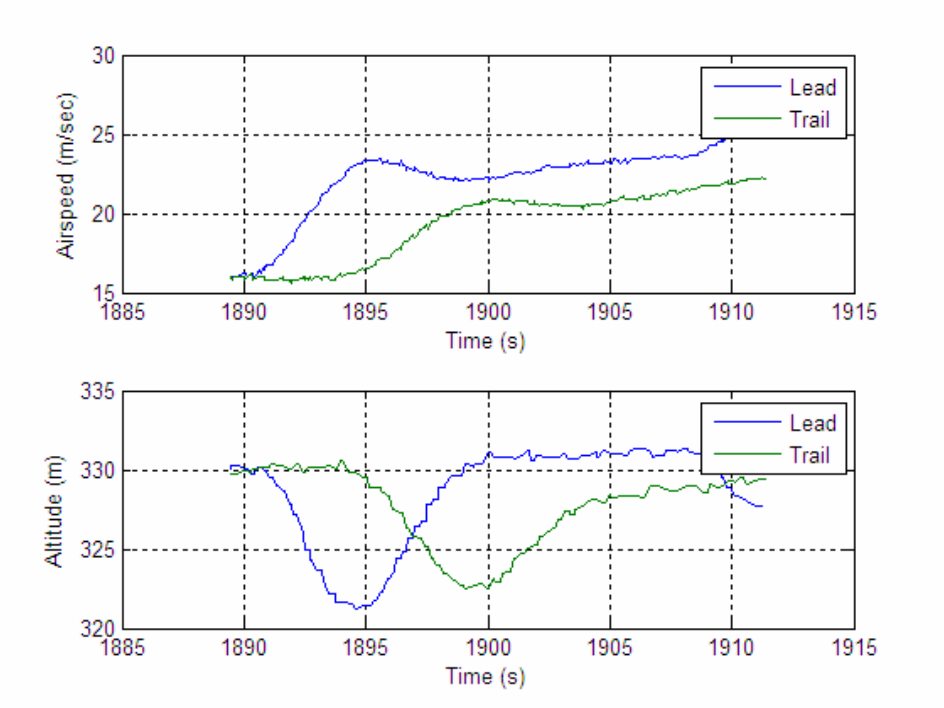

Figure 71 - Simulation #11 Time Delay: Airspeed Change

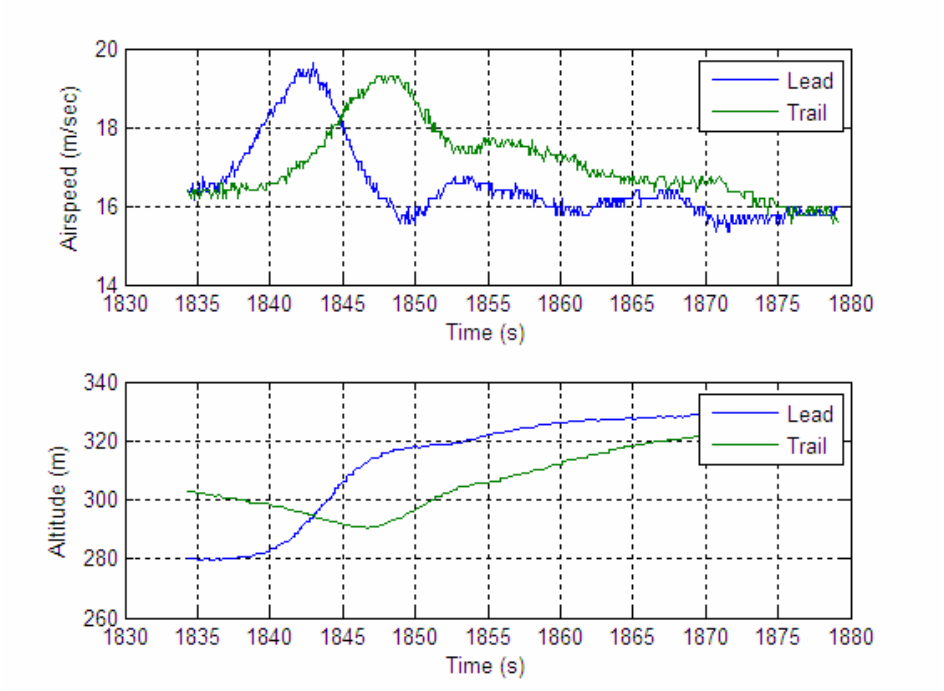

Figure 72 - Simulation #12 Time Delay: Altitude Change

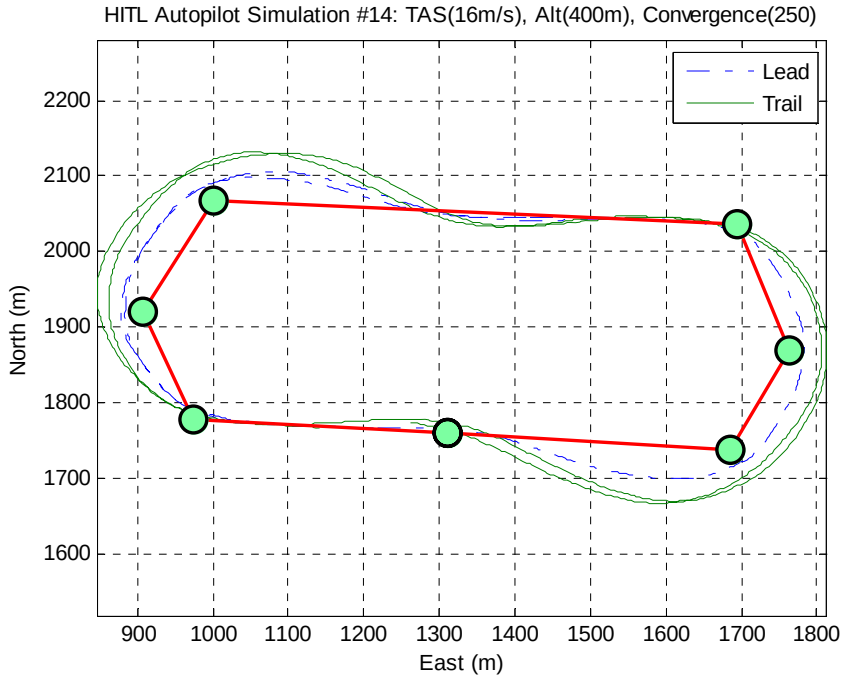

Figure 73 - Simulation #14 Airspeed 16m/s, Track Convergence 250

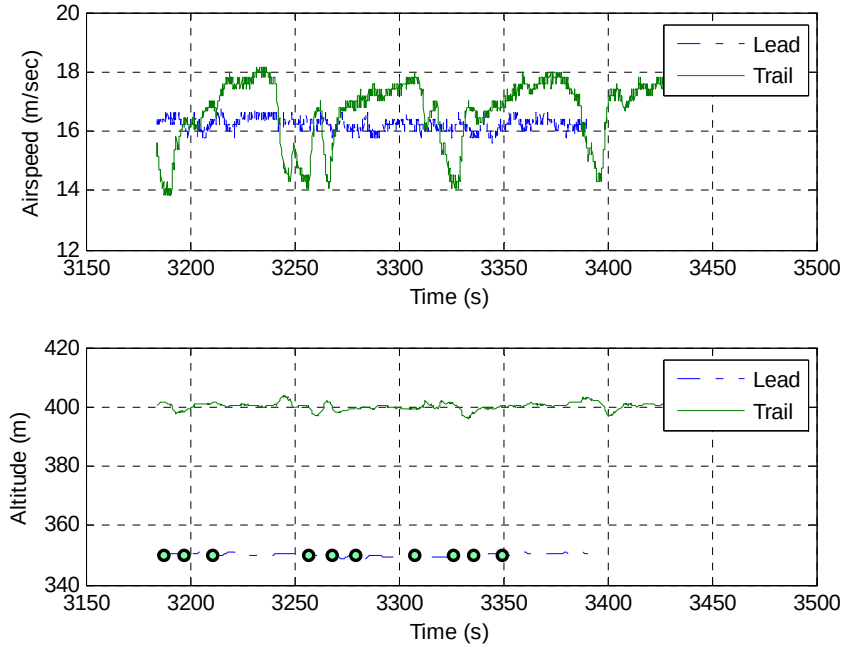

Figure 74 - Simulation #14 Airspeed 16m/s, Track Convergence 250

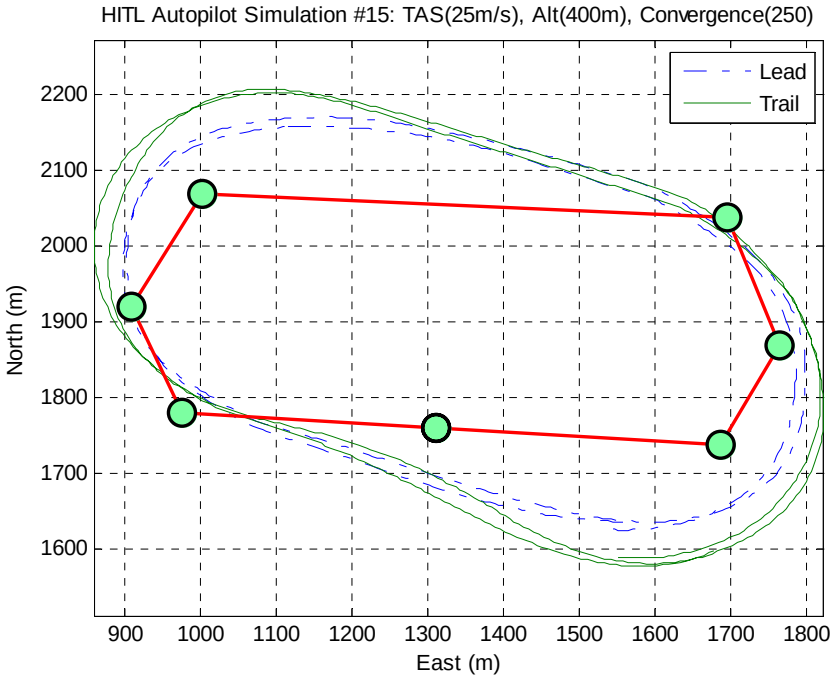

Figure 75 - Simulation #15 Airspeed 25m/s, Track Convergence 250

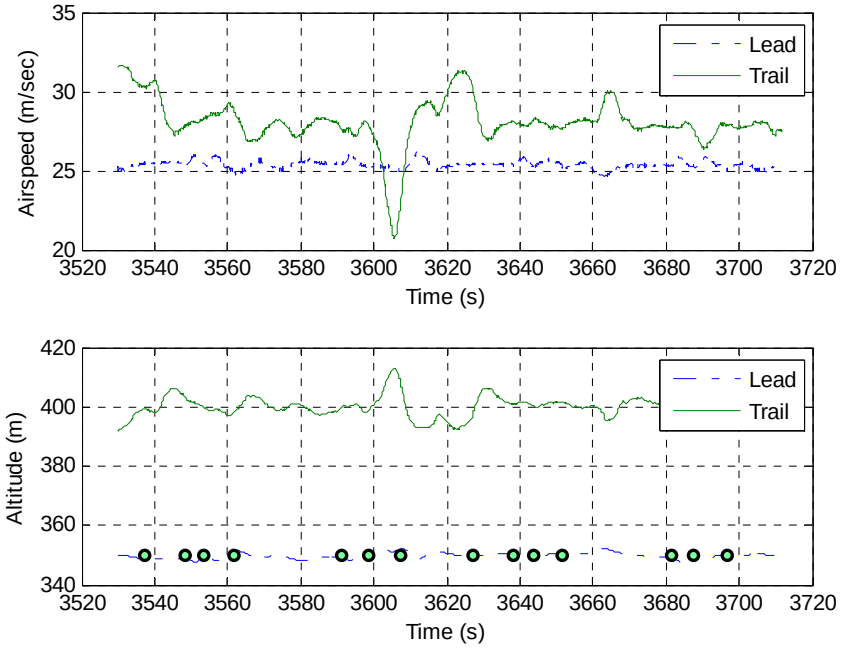

Figure 76 - Simulation #15 Airspeed 25m/s, Track Convergence 250

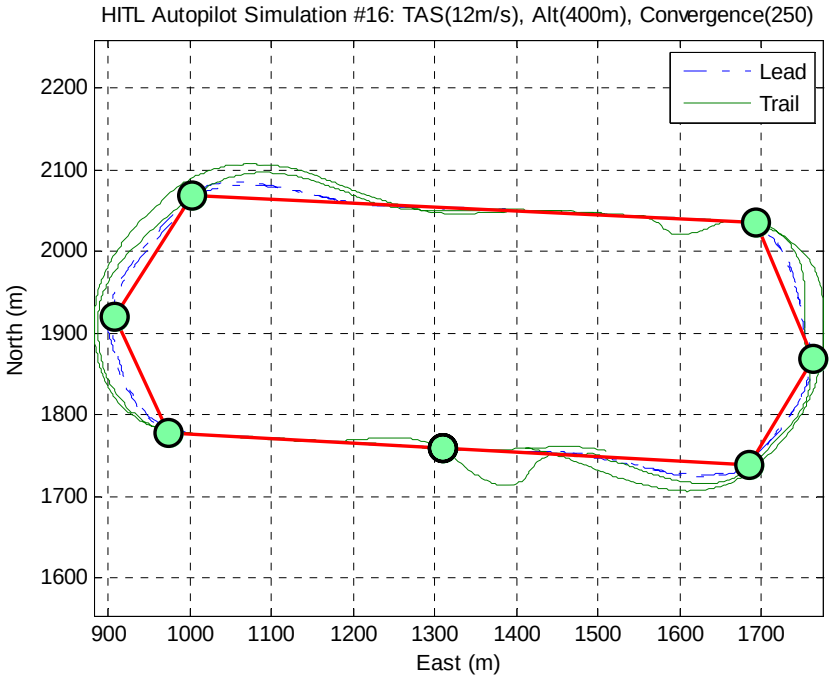

Figure 77 - Simulation #16 Airspeed 12m/s, Track Convergence 250

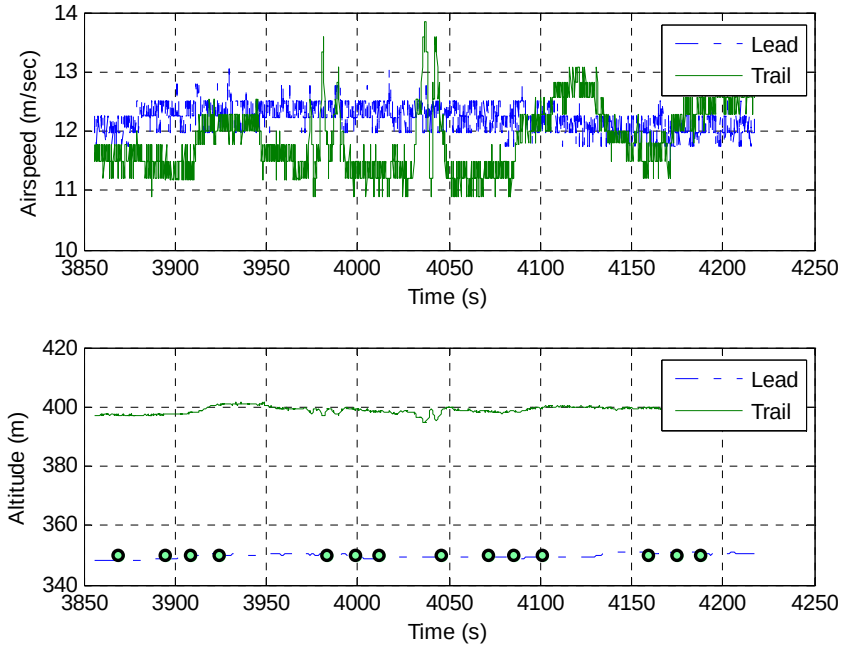

Figure 78 - Simulation #16 Airspeed 12m/s, Track Convergence 250

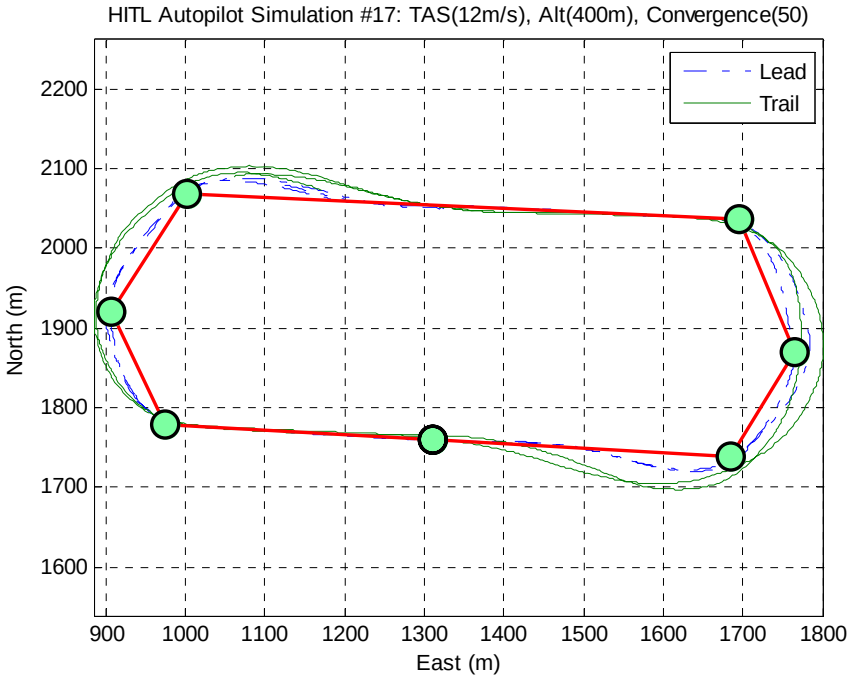

Figure 79 - Simulation #17 Airspeed 12m/s, Track Convergence 50

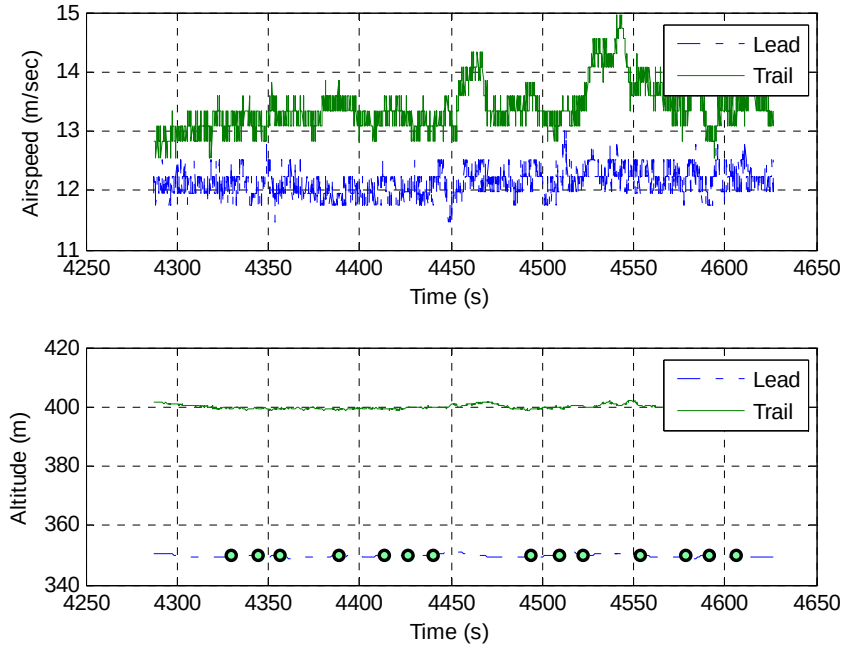

Figure 80 - Simulation #17 Airspeed 12m/s, Track Convergence 50

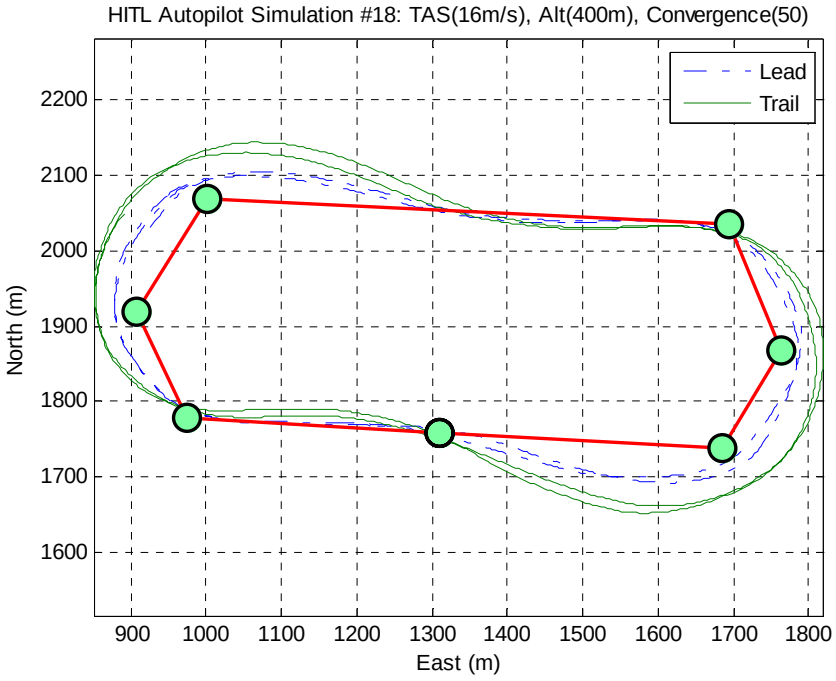

Figure 81 - Simulation #18 Airspeed 16m/s, Track Convergence 50

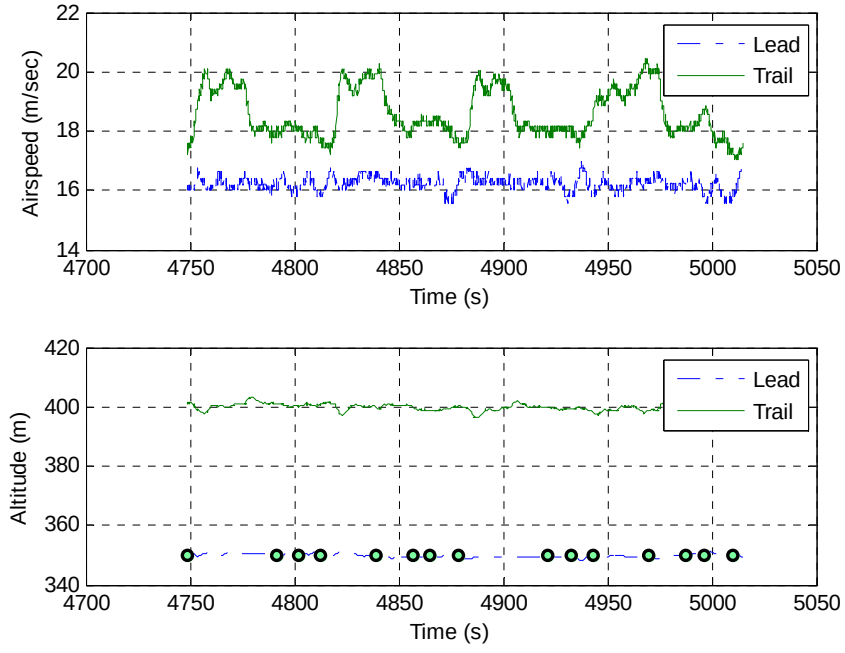

Figure 82 - Simulation #18 Airspeed 16m/s, Track Convergence 50

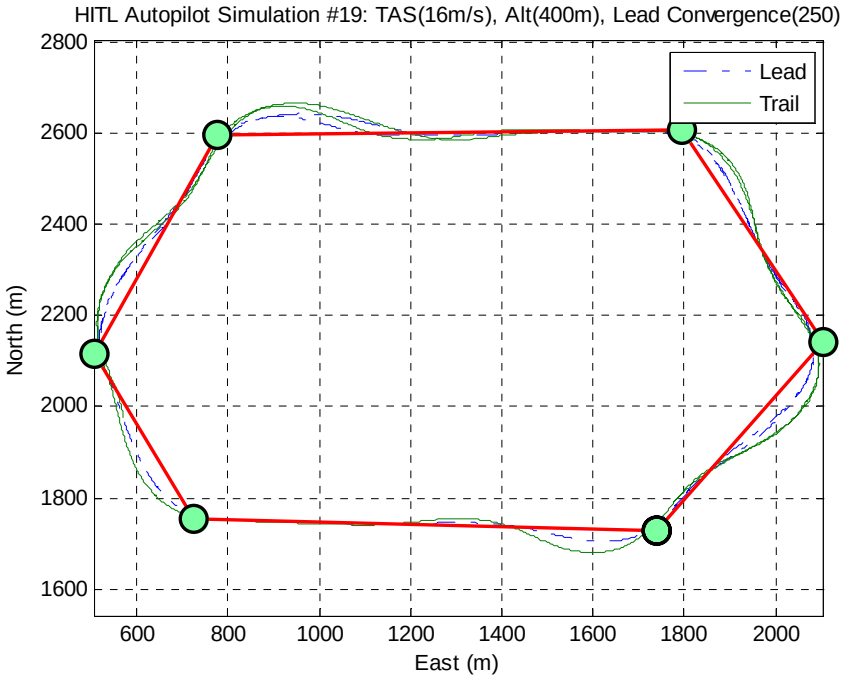

Figure 83 - Simulation #19 Airspeed 16m/s, Track Convergence 250

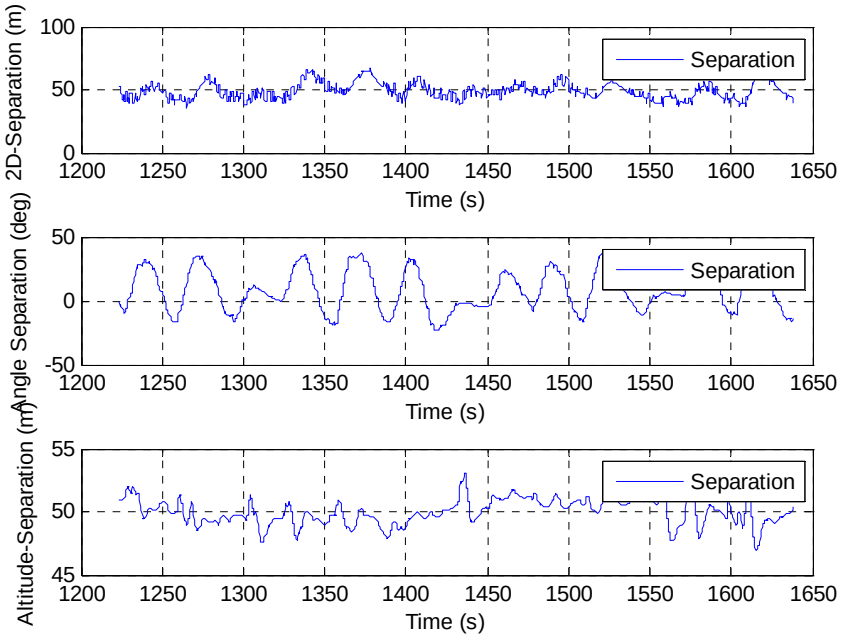

Figure 84 - Simulation #19 Airspeed 16m/s, Track Convergence 250

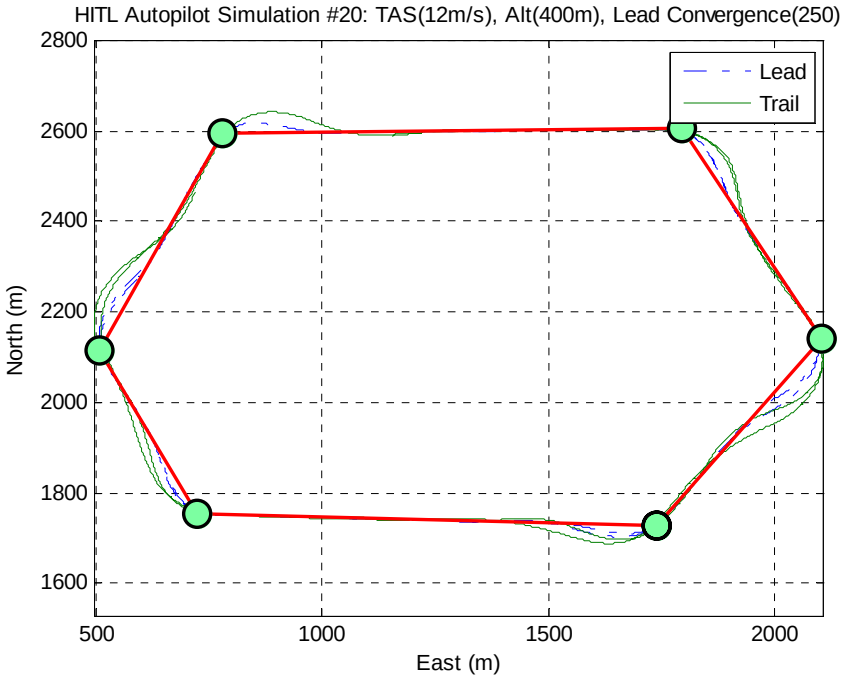

Figure 85 - Simulation #20 Airspeed 12m/s, Track Convergence 250

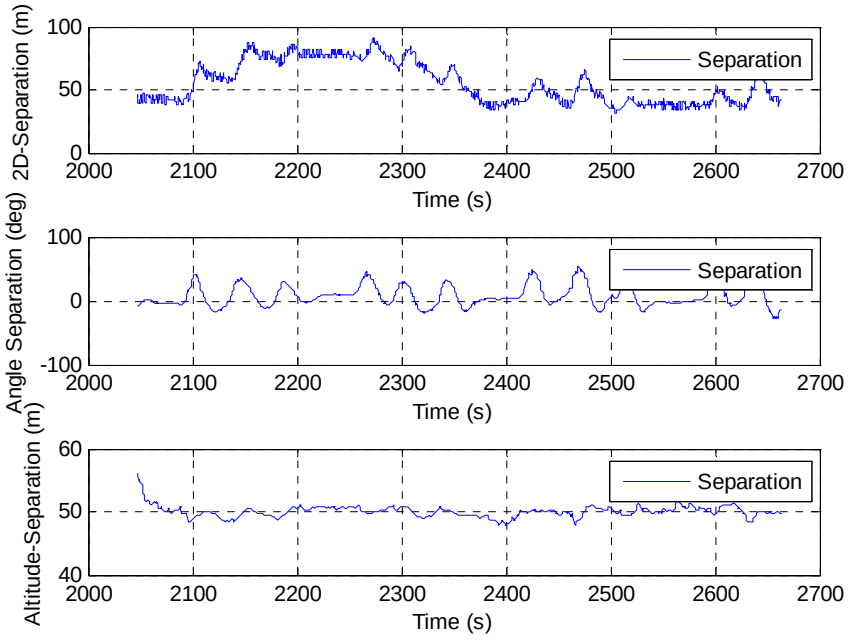

Figure 86 - Simulation #20 Airspeed 12m/s, Track Convergence 250

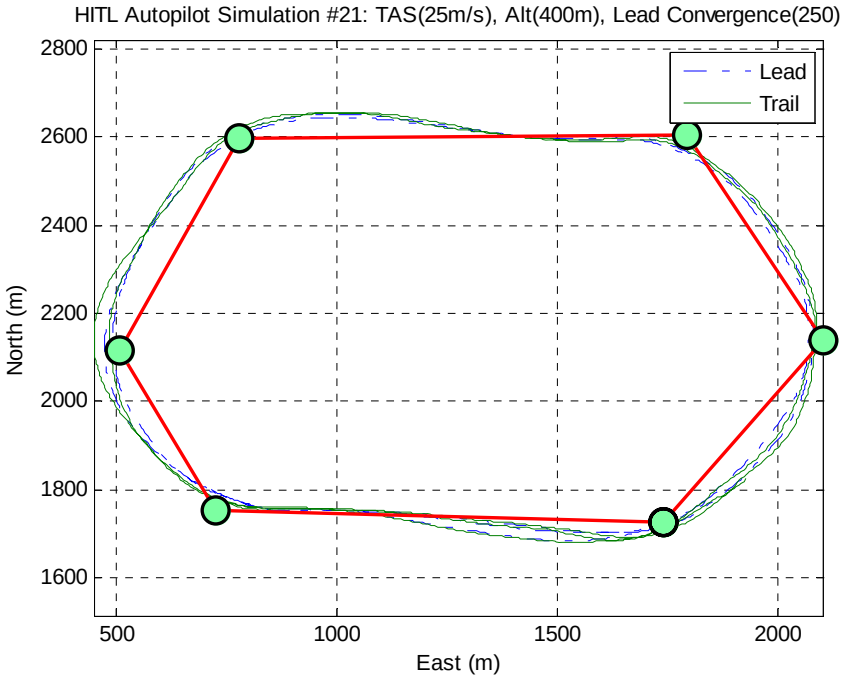

Figure 87 - Simulation #21 Airspeed 25m/s, Track Convergence 250

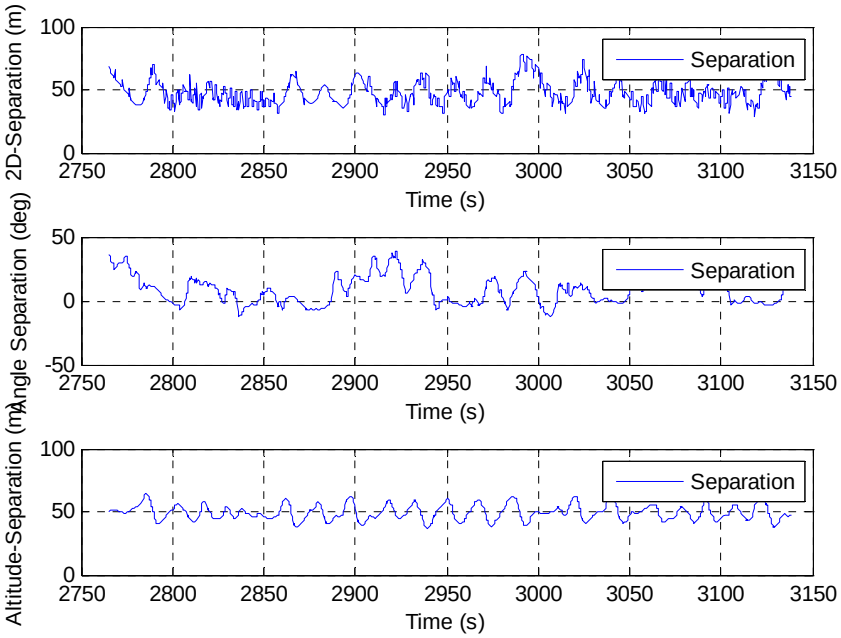

Figure 88 - Simulation #21 Airspeed 25m/s, Track Convergence 250

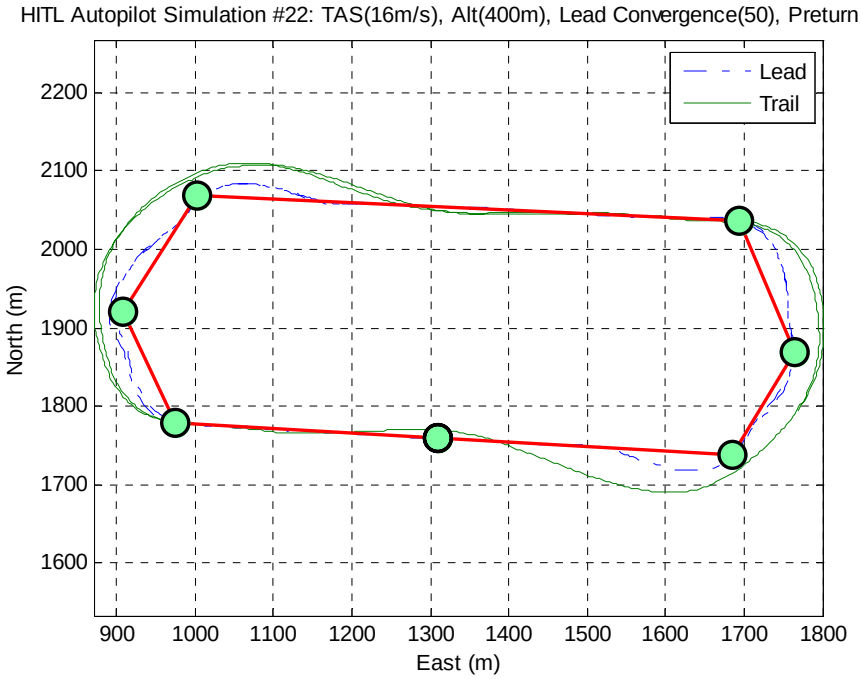

Figure 89 - Simulation #22 Airspeed 16m/s, Track Convergence 50, Preturn On

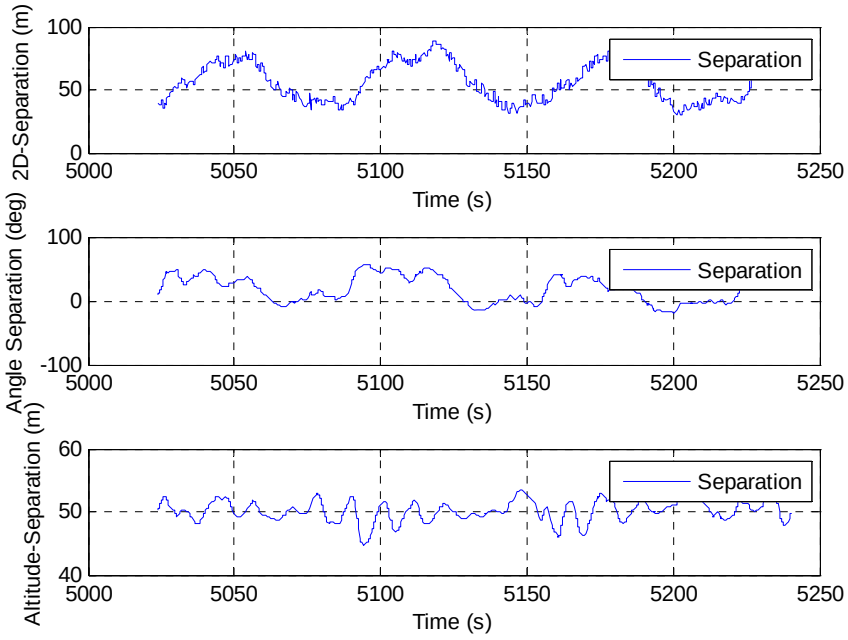

Figure 90 - Simulation #22 Airspeed 16m/s, Track Convergence 50, Preturn On

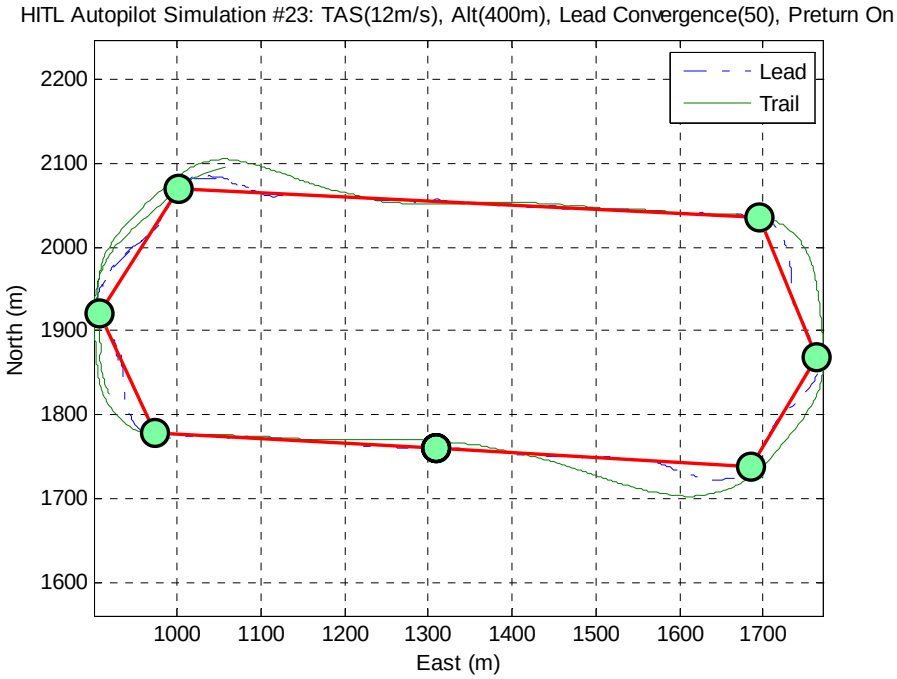

Figure 91 - Simulation #23 Airspeed 12m/s, Track Convergence 50, Preturn On

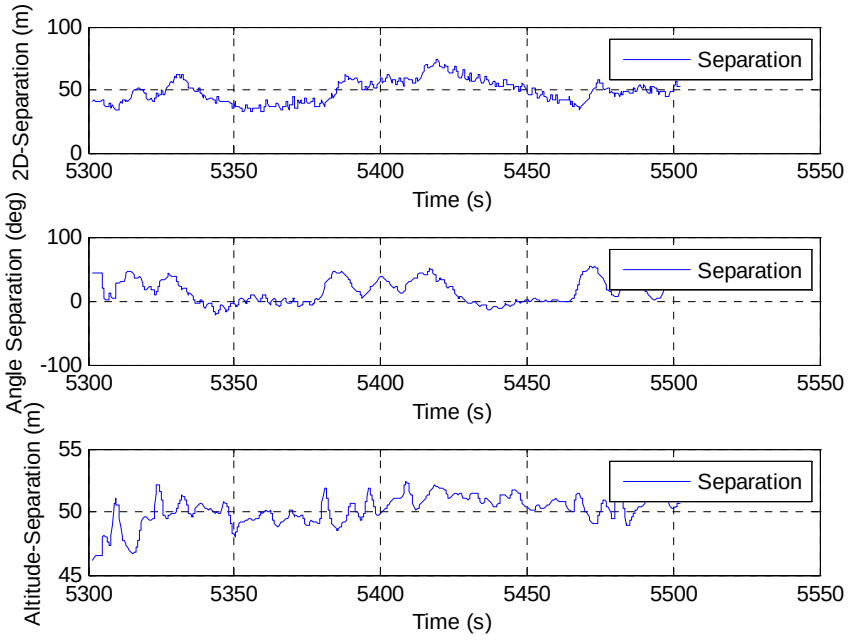

Figure 92 - Simulation #23 Airspeed 12m/s, Track Convergence 50, Preturn On

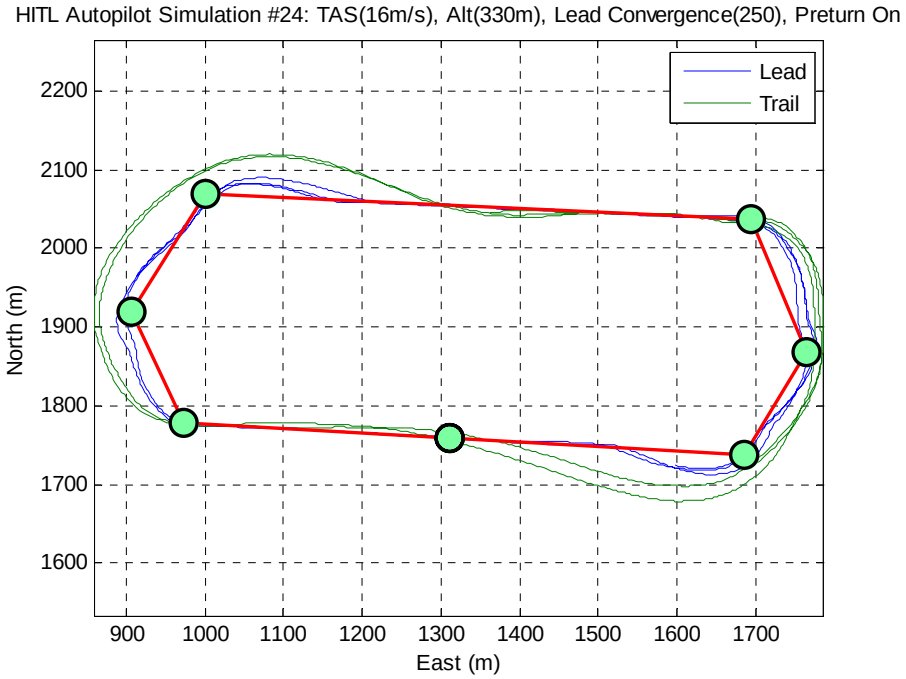

Figure 93 - Simulation #24 Airspeed 16m/s, Logarithmic Gain

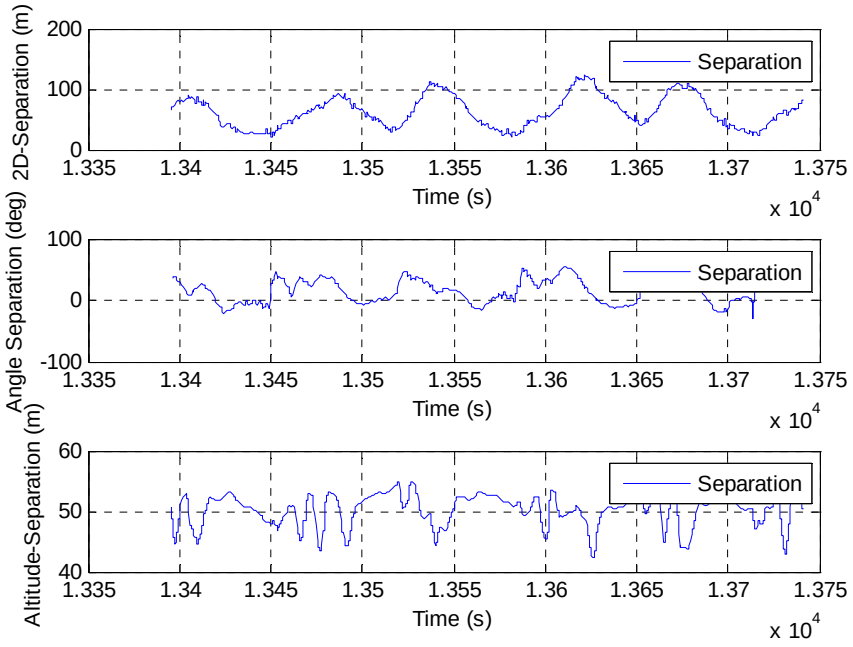

Figure 94 - Simulation #24 Airspeed 16m/s, Logarithmic Gain

Appendix B – Formation Flight Controller

```
/*****************************************************************************
Test file for piccolo communication

Programmed by: Pat McCarthy

Date: 5 May, 2006

lla2enu.h programmed by Randall Plate

*****************************************************************************/
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<string>
#include<stdio.h>
#include<cstdlib>
#include<windows.h>
#include"lla2enu.h"
#include"my_types.h"
#include "CommManager.h"
#include "Win32Serial.h"

using namespace std;

//function prototypes
void displayData(int i, int NumNets);
void InsertWaypoint(double lat, double lon, double alt, int i, int next);
void CalcRelativePosition(int NumNets);
void FormationFlight(int i, int NumNets);
void ChangeFormation(char key);
void displayTelemetry(int i);
void displayChangeInFormation(int i);
void displayFormation(int i, int NumNets);
void DoStuff(int numObst, int NumNets);
void DefineFormations();
void Calc_Dist_to_Waypoint(int i, float wpNorth, float wpEast, float alt);
void AirspeedAdjust(int i);
void ManualChange();
//Basic Variables/Arrays vital to all parts of code
CCommManager* m_pComm = NULL;       //initialize Communications Manager m_pComm
Queue_t* pQ = NULL;                         //used to see if autopilot packets exist
ENUCoord PosENU;                            //East-North-Up coordinate used for telemetry and
avoiding obstacles
ENUCoord PosENU_form;                       //East-North-Up coordinate used for formation
flight...(could be combined with PosENU)
telemetry current_telemetry[10];//holds decoded telemetry packet data for up to 10 networks
control current_control[10];          //holds decoded control packet data for up to 10 networks
```

```
//Basepoint to use for all ENU coordinates...calculated by doing lla2ecef transformation in matlab at a
lat/lon/alt near AFIT
//Note that the further the basepoint from actual position, the more error
const double Base_X = 503000;
const double Base_Y = -4884700;
const double Base_Z = 4057800;



FILE * pFile1;
FILE * pFile2;
FILE * pFile3;

//Obstacle Variables/Arrays
int distances[10][10];                  //holds distances from all networks to all obstacles
double alpha[10] = {0};                 //angles between network and obstacles
double theta[10] = {0};                 //angles between heading and obstacles...used in collision
detection
location obstacle[10];                  //hold obstacle data- allows for 10 different obstacles
bool waypoint1 = false;                 //flag to recognize when obstacle has been avoided
const int Crit_Dist = 300;          //critical distance before avoiding obstacle (meters)
const int Crit_Angle = 45;           //critical angle before avoiding obstacle (degrees)

//Formation Variables/Arrays
float reverse_cartesian[10];
float cartesian_heading[10];
formation my_formation[4];              //global array to hold predefined formation information-
allows for 4 predefined formations
int current_form=2;                             //starts formation in copositional formation
double formation_dist[10][10];      //holds distances between all networks to one another
double formation_angles[10][10];//holds angles between each network and lead network's heading (angle
off the lead's tail)
double formation_alt[10][10];       //holds formation altitude separation
double dist_to_wpoint[10];              //holds distance to desired formation waypoint
UInt8 Waypoint_cmd[10];                 //holds the index of the waypoint each network is currently
heading towards - up to 10 networks
Waypoint_t location_leader;             //holds waypoint information (Lat, Lon, Alt) of leader's
current position
int toggle_formation=0;                 //toggles formation on and off
int counter=0;                                  //sets arbitrary counter to periodically insert
formation waypoints, adjust airspeed, etc
int data=96;                                    //toggles which display you want to see...arbitrary
starting value
int leadID=562;                                 //starting lead aircraft -> Piccolo ID = 562
int trailID=565;                        //starting trail aircraft -> Piccolo ID = 565
int oldleadID;                                  //temporary variable used to switch lead/trail
double wpEast, wpNorth, head, alpha2, alt;  //temporary variables to calculate desired waypoint position
before InsertWaypoint inserts them
int lead_index;                                 //global value stores lead network's ID...avoids
having to call m_pCommGetIndexFromID(leadID) all the time
int trail_index;                        //global value stores trail network's ID...avoids having to call
m_pCommGetIndexFromID(trailID) all the time
```

```
//clears the screen
void clrscr()       {
  HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
  COORD coord = {0, 0};
  DWORD count;
  CONSOLE_SCREEN_BUFFER_INFO csbi;
  GetConsoleScreenBufferInfo(hStdOut, &csbi);
  FillConsoleOutputCharacter(hStdOut, ' ', csbi.dwSize.X * csbi.dwSize.Y, coord, &count);
  SetConsoleCursorPosition(hStdOut, coord);
}
//clears the screen


//As defined in "index.html": from SDK documentation
void NewNetwork(UInt16 NetworkID, void* Parameter)        {

}
//Needed to Initialize Networks


//Looks for and gleans data from an autopilot packet sent from a network
void LookForAutopilotData(QType* pQ, int whosData)
{
   static AutopilotPkt_t APPkts[10];
          static AutopilotCmd_t Cmd[10];
          double mins, hours;
          UInt32 i, NumNets;
          SInt32 ID;

          //look at how many networks m_pComm can see
          NumNets = m_pComm->GetNumNets();
          lead_index=m_pComm->GetIndexFromID(leadID);
          trail_index=m_pComm->GetIndexFromID(trailID);


          for(i = 0; i < NumNets; i++)
          {
                    // Don't display past 10 networks since we didn't include the space
                    if(i >= 10) break;

                    ID = m_pComm->GetIDFromIndex(i);

                    // Don't try to decode ground station packets
                    //if(ID < 1) continue;

                    //   Get the pointer to the receive queue for the autopilot stream.  Note
                    //   this pointer will persist as long as the network structure exists,
                    //   so we could just save the pointer and then we wouldn't have the
                    //   overhead of repeatedly calling this function
                    pQ = m_pComm->GetStreamRxBuffer((UInt16)ID, AUTOPILOT_STREAM);
```

```
		if(!pQ) continue;

		// Now check to see if a packet exists.  Note!!! The raw packet
		//   structure MUST persist between calls, and it MUST be unique to this
		//   network.
		if(LookForAutopilotPacket(pQ, &(APPkts[i])))
		{
			switch(APPkts[i].PktType)
			{
				case TELEMETRY:
					UserData_t telemData;
					DecodeTelemetryPacket(&(APPkts[i]), &(telemData));
					//update telemtry struct

					current_telemetry[i].Longitude = telemData.GPS.Longitude *
180.0 / 3.1415926;
					current_telemetry[i].Latitude = telemData.GPS.Latitude *
180.0 / 3.1415926;
					current_telemetry[i].Altitude = telemData.GPS.Altitude;
					current_telemetry[i].Velocity = telemData.GPS.Speed;


					//convert lla data to enu
					PosENU.lla2enu(current_telemetry[i].Latitude
*3.1415926/180,
							current_telemetry[i].Longitude
*3.1415926/180,
							current_telemetry[i].Altitude,
							Base_X, Base_Y, Base_Z);

					current_telemetry[i].East = PosENU.GetEast();
					current_telemetry[i].North = PosENU.GetNorth();
					current_telemetry[i].Up = PosENU.GetUp();
					current_telemetry[i].Hours = telemData.GPS.hours;
					current_telemetry[i].Minutes = telemData.GPS.minutes;
					current_telemetry[i].Seconds = telemData.GPS.seconds;

					//Getting leader's telemetry
					if(i==lead_index) {
						location_leader.Lat=current_telemetry[i].Latitude;
						location_leader.Alt=current_telemetry[i].Altitude;
						location_leader.Lon=current_telemetry[i].Longitude;
					}


					//display the data
					fprintf(pFile1,"\n %i	%7i	%f	%f	%f",leadID,
current_telemetry[lead_index].Time, current_telemetry[lead_index].Latitude,
current_telemetry[lead_index].Longitude, current_telemetry[lead_index].Altitude);
					fprintf(pFile2,"\n %i	%7i	%f	%f	%f",trailID,
current_telemetry[trail_index].Time, current_telemetry[trail_index].Latitude,
current_telemetry[trail_index].Longitude, current_telemetry[trail_index].Altitude);
					fprintf(pFile3,"\n%7u	%7u	%7f	%7f	%7f",
current_telemetry[lead_index].Time, current_telemetry[trail_index].Time,
```

```
formation_dist[trail_index][lead_index], formation_angles[trail_index][lead_index],
formation_alt[trail_index][lead_index]);
                                        displayData(whosData, NumNets);
                                        break;
                                case CONTROL_DATA:
                                        UserData_t controlData;
                                        float gyroBias[3], controls[10];
                                        DecodeControlDataPacket(&(APPkts[i]), &(controlData),
gyroBias, controls);
                                        //update telemetry struct
                                        current_telemetry[i].Time = controlData.SystemTime;
                                        current_control[i].BankAngle = controlData.Euler[0] *
180/3.1415926;
                                        current_control[i].Heading = controlData.Euler[2] *
180/3.1415926;
                                                //Euler[0] = Roll, Euler[1] = Pitch, Euler[2] = Yaw
                                        current_control[i].RollRate = controlData.Gyro[0] *
180/3.1415926;
                                        current_control[i].PitchRate = controlData.Gyro[1] *
180/3.1415926;
                                        current_control[i].YawRate = controlData.Gyro[2] *
180/3.1415926;
                                        current_control[i].AirSpeed = controlData.TAS;
                                        current_control[i].Pdynamic = controlData.Pdynamic;

                                        current_control[i].Aileron = controls[0] * 180/3.1415926;
                                        current_control[i].Elevator = controls[1] * 180/3.1415926;
                                        current_control[i].Throttle = controls[2];
                                        current_control[i].Rudder = controls[3] * 180/3.1415926;
                                        //convert GPS seconds into hours, minutes, and seconds
                                        hours = controlData.SystemTime / 3600000.0;
                                        current_control[i].Hours = hours;
                                        mins = (hours - (double)current_control[i].Hours) * 60;
                                        current_control[i].Minutes = mins;
                                        current_control[i].Seconds = (mins -
(double)current_control[i].Minutes) * 60;
                                        displayData(whosData, NumNets);
                                        break;
                                case WAYPOINT: //This can give you the individual waypoint
information
                                        break;
                                case WAYPOINT_LIST:
                                        break;
                                case TRACK:
                                        break;
                                case AUTOPILOT_COMMAND:
                                        AutopilotCmd_t Cmd[3];
                                        Waypoint_cmd[i] =
DecodeAutopilotControlPacket(&(APPkts[i]), &Cmd[i]);
                                        //This returns Waypoint_cmd[i] as the waypoint the Piccolo is
currently heading towards
                                        displayData(whosData, NumNets);
                                        break;
```

```
                        }
                }
        }
}
// LookForAutopilotData


//Displays Instructions
void displayData(int i, int NumNets) {
        clrscr();
        printf("Instructions");
        printf("\nPress a Number to see Individual Piccolo Data (1,2...)");
        printf("\nPress 'T' to see Telemetry Data");
        printf("\nPress 'F' to see Formation Data");
        printf("\nPress 'C' to Change Formation Characteristics");
        printf("\nPress 'M' to Manually Change Lead's Commands");
        printf("\nPress 'X' to Exit");

        printf("\nCurrent Piccolo ID = %i", m_pComm->GetIDFromIndex(i));
        printf("\nCurrently heading towards Waypoint: %i", Waypoint_cmd[i]);

        if(data==0)
                displayTelemetry(i);
        if(data==1)
                displayFormation(i, NumNets);
        if(data==2)
                displayChangeInFormation(i);
        if(data==3)
                ManualChange();
        else;
}
//displayData


//Displays Network Telemetry and Control Information
void displayTelemetry(int i) {
        printf("\nSystem Time: %u", current_telemetry[i].Time);
        printf("\nTelemetry Packet Data  : %i", current_telemetry[i].Hours);
        printf(":%i", current_telemetry[i].Minutes);
        printf(":%f", current_telemetry[i].Seconds);
        printf("\nLatitude (deg)         : %f", current_telemetry[i].Latitude);
        printf("               East: %f", current_telemetry[i].East);
        printf("\nLongitude (deg)        : %f", current_telemetry[i].Longitude);
        printf("               North: %f", current_telemetry[i].North);
        printf("\nAltitude (m)           : %f", current_telemetry[i].Altitude);
        printf("               Up: %f", current_telemetry[i].Up);
        printf("\nGround Speed           : %f", current_telemetry[i].Velocity);
        printf("\nAir Speed              : %f", current_control[i].AirSpeed);
        //print current control data
        printf("\n\nControl Packet Data  : %i", current_control[i].Hours);
        printf(":%i", current_control[i].Minutes);
        printf(":%f", current_control[i].Seconds);
        printf("\nHeading                : %f", current_control[i].Heading);
```

```
		printf("		Aileron (deg)	: %f", current_control[i].Aileron);
		printf("\nBank Angle		: %f", current_control[i].BankAngle);
		printf("		Elevator (deg)	: %f", current_control[i].Elevator);
		printf("\nRoll Rate		: %f", current_control[i].RollRate);
		printf("		Throttle (percent): %f", current_control[i].Throttle*100);
		printf("\nPitch Rate		: %f", current_control[i].PitchRate);
		printf("		Rudder (deg)	: %f", current_control[i].Rudder);
		printf("\nYaw Rate		: %f", current_control[i].YawRate);
		//print current surface deflections
}
//displayTelemetry

//Displays Formation Characteristics
void displayFormation(int i, int NumNets) {
			//Prints Formation Data
			int m=m_pComm->GetIndexFromID(leadID);
			printf("\nCurrent Lead Aircraft: %i",leadID);
			printf("\n\nAngle between heading and obstacle: %f", theta[i]);
			printf("\nNumber of Piccolo's seen: %i", NumNets-1);
			printf("\nRadial Separation from Leader: %f",formation_dist[i][m]);
			printf("\nAngle Separation from Leader: %f", formation_angles[i][m]);
			printf("\nAltitude Separation from Leader: %f", formation_alt[i][m]);
			printf("\nRatio of Trail Airspeed to Lead Airspeed: %f",
current_control[trail_index].AirSpeed/current_control[lead_index].AirSpeed);
			if(i != m) {
				printf("\nDistance Behind Desired Waypoint: %f",dist_to_wpoint[i]);
			}
			//only print distance to waypoint for trailing aircraft
}
//displayFormation


//Displays Changing Formation Instructions and Calls ChangeFormation
void displayChangeInFormation(int i) {

			printf("\nPress 'S' to maintain Formation but switch Leader");
			printf("\nPress 'L' for Line Formation");
			printf("\nPress 'V' for V Formation");
			printf("\nPress 'D' for Above Formation");
			printf("\nPress 'W' for 90 Degree Wing Formation");
			printf("\nPress 'O' to turn Formation Flight Off");

			char key=getch();
			ChangeFormation(key);
			printf("\nPress any key to continue...");
			getch();
			data=90;
}
//displayChangeInFormation



void ManualChange() {
```

```
		printf("\nPress 'S' to Speed Up");
		printf("\nPress 'D' to Slow Down");
		printf("\nPress 'H' to Command Higher Altitude (50 m)");
		printf("\nPress 'L' to Command Lower Altitude (50 m)");
		printf("\nPress 'T' to Track New Waypoint By Index");

		static AutopilotLoopCmd_t altCom;
		static AutopilotLoopCmd_t speedCom;
		float cmd_speed;
		char key=getch();
		switch(key) {
			case 's':
				cmd_speed=current_control[lead_index].Pdynamic*1.1;
				speedCom.Loop = 0;			//command a dynamic pressure
				speedCom.Control = 1;	//turn ap_loop_cmd on
				speedCom.Value = (cmd_speed);//assign the commanded value
				m_pComm->SendAutopilotLoopControlPacket(leadID,
&(speedCom));//send the command
				break;
			case 'd':
				cmd_speed=current_control[lead_index].Pdynamic*0.9;
				speedCom.Loop = 0;			//command a dynamic pressure
				speedCom.Control = 1;	//turn ap_loop_cmd on
				speedCom.Value = (cmd_speed);//assign the commanded value
				m_pComm->SendAutopilotLoopControlPacket(leadID,
&(speedCom));//send the command
				break;
			case 'h':
				altCom.Loop=1;
				altCom.Control=1;
				altCom.Value=current_telemetry[lead_index].Altitude+50;
				m_pComm->SendAutopilotLoopControlPacket(leadID,
&(altCom));//send the command
				break;
			case 'l':
				altCom.Loop=1;
				altCom.Control=1;
				altCom.Value=current_telemetry[lead_index].Altitude-50;
				m_pComm->SendAutopilotLoopControlPacket(leadID,
&(altCom));//send the command
				break;
			case 't':
				printf("\nEnter New Waypoint Command");
				int track=getch();
				m_pComm->SendTrackCommandPacket(leadID, track, true);  //send
command to head to new waypoint
				break;
		}
		printf("\nPress any key to continue...");
		getch();
		data=90;
}
```

```
//Changes Formation Characteristics
void ChangeFormation(char key) {
	switch(key) {
		case 'l':
			current_form=0;
			printf("\n\nFormation Changed to Line");
			break;
		case 'v':
			current_form=1;
			printf("\n\nFormation Changed to V");
			break;
		case 'd':
			printf("\n\nFormation Changed to Above");
			current_form=2;
			break;
		case 's':
			oldleadID=leadID;
			leadID=trailID;
			trailID=oldleadID;
			printf("\n\nLead Aircraft Switched to: %i",leadID);
			printf("\nTrail Aircraft is now: %i",trailID);
		case 'w':
			current_form=3;
			printf("\n\nFormation Changed to 90 Degree Wing Formation");
			break;
		case 'o':
			toggle_formation++;
			printf("\n\nFormation Toggled");
			break;
		default:
			break;
	}
}
//ChangeFormation



//Defines Predetermined Formations
void DefineFormations() {

	//Formation 0 is Line
	my_formation[0].radial_sep=30;
	my_formation[0].z_sep=0;
	my_formation[0].bearing=0;

	//Formation 1 is V
	my_formation[1].radial_sep=30;
	my_formation[1].z_sep=0;
	my_formation[1].bearing=45;

	//Formation 2 is Concurrent Position (SIMULATION ONLY!!!)
```

```
		my_formation[2].radial_sep=0;
		my_formation[2].z_sep=50;
		my_formation[2].bearing=0;

		//Formation 3 is 90 degree Wing Position
		my_formation[3].radial_sep=30;
		my_formation[3].z_sep=0;
		my_formation[3].bearing=90;


}
//DefineFormations



//Calculates distance to desired waypoint
void Calc_Dist_to_Waypoint(int i, float wpNorth, float wpEast, float alt) {//replace these with wpEast
		dist_to_wpoint[i]=sqrt((current_telemetry[i].North-wpNorth)*(current_telemetry[i].North-
wpNorth)+(current_telemetry[i].East-wpEast)*(current_telemetry[i].East-wpEast));
}
//Calc_Dist_to_Waypoint

//Avoids Obstacles, Formation Flight, etc
void DoStuff(int numObst, int NumNets)
{
		for(int i=0; i<NumNets; i++) {
			// Don't display past 10 networks since we didn't include the space
			if(i >= 10) break;

			//AvoidObstacles(numObst,i);
			if(toggle_formation %2 ==0) {
				FormationFlight(i, NumNets);
			}//FormationFlight can be toggled on and off
		}
}
//DoStuff



//Initiates formation flight if it sees more than 2 networks and toggle_formation is turned on
void FormationFlight(int i, int NumNets) {
		static AutopilotLoopCmd_t altCom;
		int ID=m_pComm->GetIDFromIndex(i);
		bool too_close=false;
		counter=counter+1;
		if(ID== trailID && counter % 50 == 0) {	//send new waypoint to trail aircraft periodically
			float wpEast, wpNorth, head, alpha2, alt;
			CalcRelativePosition(NumNets);
			head=current_control[lead_index].Heading;
			alpha2 = 90 - head;

			alt=current_telemetry[lead_index].Up+my_formation[current_form].z_sep;
```

```
		wpNorth=current_telemetry[lead_index].North+my_formation[current_form].radial_sep*sin((reve
rse_cartesian[lead_index]-my_formation[current_form].bearing)*3.14159265359/180);

		wpEast=current_telemetry[lead_index].East+my_formation[current_form].radial_sep*cos((revers
e_cartesian[lead_index]-my_formation[current_form].bearing)*3.14159265359/180);
			PosENU_form.enu2lla(wpEast, wpNorth, alt, Base_X, Base_Y, Base_Z);


			altCom.Loop=1;
			altCom.Control=1;

		altCom.Value=current_telemetry[lead_index].Altitude+my_formation[current_form].z_sep;
			m_pComm->SendAutopilotLoopControlPacket(trailID, &(altCom));//send the command


			if(formation_dist[lead_index][trail_index]-my_formation[current_form].radial_sep>10) {
//if you're more than 25 meters of the desired formation position, insert waypoint


//exactly at that desired formation position
				InsertWaypoint(PosENU_form.GetLat(),
PosENU_form.GetLong(),PosENU_form.GetAlt(), trail_index, 0);
			}
			else m_pComm->SendTrackCommandPacket(trailID, Waypoint_cmd[trail_index], true);
		//send command to head to new waypoint
				//InsertWaypoint(current_telemetry[lead_index].Latitude,
current_telemetry[lead_index].Longitude, current_telemetry[lead_index].Altitude, trail_index, 0);


//if you're too close to desired formation position, insert waypoint at the lead's aircraft


//Piccolo will also be slowing down simultaneously
			Calc_Dist_to_Waypoint(i,wpNorth, wpEast, alt);
			if(counter%20==0) //Adjust trail AC airspeed periodically
				AirspeedAdjust(i);

		}
}
//FormationFlight


//Adjusts trail AC airspeed using extremely rough proportional control (further separation => greater trail
airspeed)
void AirspeedAdjust(int i) {
		float cmd_speed, speed_ratio;
		static AutopilotLoopCmd_t speedCom;
		/*

		//This is using the "stop and go" method of gain tuning based on formation separation
```

```
	if(dist_to_wpoint[i]- my_formation[current_form].radial_sep> 300 &&
formation_dist[lead_index][trail_index] >= my_formation[current_form].radial_sep) { //Need to add
something about closing rate
			cmd_speed=current_control[lead_index].Pdynamic*1.75;
			speedCom.Loop = 0;			//command a dynamic pressure
			speedCom.Control = 1;	//turn ap_loop_cmd on
			speedCom.Value = (cmd_speed);//assign the commanded value
			m_pComm->SendAutopilotLoopControlPacket(trailID, &(speedCom));//send the
command
	}
	//this isn't right yet?? or is it???
	if(dist_to_wpoint[i]-my_formation[current_form].radial_sep > 100 && dist_to_wpoint[i]-
my_formation[current_form].radial_sep < 300 && formation_dist[lead_index][trail_index] >=
my_formation[current_form].radial_sep) {
			cmd_speed=current_control[lead_index].Pdynamic*1.5;
			speedCom.Loop = 0;			//command a dynamic pressure
			speedCom.Control = 1;	//turn ap_loop_cmd on
			speedCom.Value = (cmd_speed);//assign the commanded value
			m_pComm->SendAutopilotLoopControlPacket(trailID, &(speedCom));//send the
command
	}
	if(dist_to_wpoint[i]-my_formation[current_form].radial_sep >50 && dist_to_wpoint[i]-
my_formation[current_form].radial_sep < 100 && formation_dist[lead_index][trail_index] >=
my_formation[current_form].radial_sep) {
			if(fabs(formation_angles[trail_index][lead_index])-
my_formation[current_form].bearing>20)
				cmd_speed=current_control[lead_index].Pdynamic*0.7;
			if(fabs(formation_angles[trail_index][lead_index])-
my_formation[current_form].bearing<20)
				cmd_speed=current_control[lead_index].Pdynamic*1.2;
			else cmd_speed=current_control[lead_index].Pdynamic*1.2;
			//Must also be within proper angle to increase speed

			speedCom.Loop = 0;			//command a dynamic pressure
			speedCom.Control = 1;	//turn ap_loop_cmd on
			speedCom.Value = (cmd_speed);//assign the commanded value
			m_pComm->SendAutopilotLoopControlPacket(trailID, &(speedCom));//send the
command
	}
	if(dist_to_wpoint[i]-my_formation[current_form].radial_sep >40 && dist_to_wpoint[i]-
my_formation[current_form].radial_sep < 50 && formation_dist[lead_index][trail_index] >=
my_formation[current_form].radial_sep) {
			if(fabs(formation_angles[trail_index][lead_index])-
my_formation[current_form].bearing>20)
				cmd_speed=current_control[lead_index].Pdynamic*0.7;
			if(fabs(formation_angles[trail_index][lead_index])-
my_formation[current_form].bearing<20)
				cmd_speed=current_control[lead_index].Pdynamic*1.1;
			else cmd_speed=current_control[lead_index].Pdynamic*1.1;
			speedCom.Loop = 0;			//command a dynamic pressure
			speedCom.Control = 1;	//turn ap_loop_cmd on
			speedCom.Value = (cmd_speed);//assign the commanded value
```

```
                m_pComm->SendAutopilotLoopControlPacket(trailID, &(speedCom));//send the
command
        }
        if(dist_to_wpoint[i]-my_formation[current_form].radial_sep < 40 &&
formation_dist[lead_index][trail_index] >= my_formation[current_form].radial_sep) {
                cmd_speed=current_control[lead_index].Pdynamic*0.6;
                speedCom.Loop = 0;              //command a dynamic pressure
                speedCom.Control = 1;   //turn ap_loop_cmd on
                speedCom.Value = (cmd_speed);//assign the commanded value
                m_pComm->SendAutopilotLoopControlPacket(trailID, &(speedCom));//send the
command
        }
        if(formation_dist[lead_index][trail_index] < my_formation[current_form].radial_sep) {
                cmd_speed=current_control[lead_index].Pdynamic*0.56;  // slow to 3/4 speed if about to
go ahead, and insert a new waypoint where the lead aircraft is (so no sudden changes)
                speedCom.Loop = 0;              //command a dynamic pressure
                speedCom.Control = 1;   //turn ap_loop_cmd on
                speedCom.Value = (cmd_speed);//assign the commanded value
                m_pComm->SendAutopilotLoopControlPacket(trailID, &(speedCom));//send the
command
        }
        else;*/

        //
        //this is using the logarithmic gain, as opposed to the "stop and go" method above
        //

        if(fabs(formation_angles[trail_index][lead_index])-my_formation[current_form].bearing>20)
                speed_ratio=0.8;
        else speed_ratio=-0.16+0.3*log(dist_to_wpoint[i]+2.7182818);

        //this is the actual gain equation - needs to be tuned better

        cmd_speed=current_control[lead_index].Pdynamic*speed_ratio*speed_ratio;
        speedCom.Loop=0;
        speedCom.Control=1;
        speedCom.Value=cmd_speed;
        m_pComm->SendAutopilotLoopControlPacket(trailID, &(speedCom));
        //Need to add something about "If you're too far away AND the angle is within range..."
}
//AirspeedAdjust


//Calculates relative position between all network's in ENU coordinates
void CalcRelativePosition(int NumNets) {
        //int NumNets=m_pComm->GetNumNets();
        double posn[10], pose[10], alt[10];
        float theta2;

        for(int m=0;m<NumNets;m++) {
                for(int n=0;n<NumNets;n++) {
                        if(m_pComm->GetIDFromIndex(m) > 560 && m_pComm-
>GetIDFromIndex(n)> 560) {
```

```
                                        //Should make it so it only switches actual Piccolo's
                                        posn[n]=current_telemetry[n].North;
                                        pose[n]=current_telemetry[n].East;
                                        alt[n]=current_telemetry[n].Altitude;
                                        formation_dist[m][n]=sqrt((posn[m]-posn[n])*(posn[m]-
posn[n])+(pose[m]-pose[n])*(pose[m]-pose[n]));
                                        theta2=180/3.14159265359*atan2((posn[m]-posn[n]),(pose[m]-
pose[n]));
                                        if(theta2<0) // atan2 returns 0_to_pi, 0_to_-pi
                                                theta2=360+theta2;
                                        //this returns the cartesian angle between the lead (n) and trail (m)
aircraft

                                        if(current_control[m].Heading>=0 && current_control[m].Heading
<=90)
                                                cartesian_heading[m]=90-current_control[m].Heading;
                                        if(current_control[m].Heading>90 && current_control[m].Heading
<=360)
                                                cartesian_heading[m]=450-current_control[m].Heading;
                                        //This gives the cartesian_heading of the aircraft m...should it be 'n'?

                                        if(cartesian_heading[m]>=180)
                                                reverse_cartesian[m]=cartesian_heading[m]-180;
                                        else if(cartesian_heading[m]<180)
                                                reverse_cartesian[m]=cartesian_heading[m]+180;
                                        formation_angles[m][n]=reverse_cartesian[m]-theta2;
                                        if(reverse_cartesian[m]-theta2>=180)
                                                formation_angles[m][n]=reverse_cartesian[m]-theta2-360;
                                        else if(reverse_cartesian[m]-theta2<= -180)
                                                formation_angles[m][n]=reverse_cartesian[m]-theta2+360;
                                        if(m==n)
                                                formation_angles[m][n]=0;
                                        formation_alt[m][n]=alt[m]-alt[n];
                                }
                        }
                }
}
//CalcRelativePosition


//Sends waypoint packet to add waypoint to flight path at given location
void InsertWaypoint(double lat, double lon, double alt, int whosData, int next)
{
        UInt16 ID = m_pComm->GetIDFromIndex(whosData);
        FPPoint_t WPInfo;
         Waypoint_t location;
        location.Lat = lat;
        location.Lon = lon;
        location.Alt = alt;
        //location of new waypoint

        WPInfo.Point = location;                                        //Assigns lat/lon/alt
        WPInfo.Next = Waypoint_cmd[lead_index];                         //Next waypoint index
```

```
	WPInfo.PreTurn = 1;                                    //1 for preturn on, 0 for off
	WPInfo.Direction=0;                                    //1 for orbit right, 0 for left

	//float OrbitRadius;        //!< Radius of the orbit in meters
	//UInt16 OrbitTime;         //!< Seconds spent in the orbit
	//WPInfo.OrbitRadius=50;
	//WPInfo.OrbitTime=50;   //If you want a trail to orbit a lead aircraft, this could be turned on


	m_pComm->SendWaypointPacket(ID, &(WPInfo), 99);        //send new waypoint
	m_pComm->SendTrackCommandPacket(ID, 99, true);         //send command to head to new
waypoint

	//        third parameter indicates if the vehicle should fly to the waypoint along the
	//  preceding track segment, or if it should go directly to the waypoint, using its
	//        current position as the starting point. Set to TRUE to go directly to the waypoint.


}
//InsertWaypoint


//Main Program
int main()
{
	int numObst=1;

	//Open files that will store logged formation data
	pFile1 = fopen ("Log1.txt","w");
	pFile2 = fopen ("Log2.txt","w");
	pFile3 = fopen ("Log3.txt","w");




	//create CCommManager object to communicate with Piccolo
	// 129.92.5.112 is the IP address of the operator interface computer (Fly B)
	m_pComm = new CCommManager(0, 57600, "129.92.5.112:2000", 0);
	//m_pComm = new CCommManager(1,"",2000);
	// This doesn't work, but this should be the way to run code on same computer
	// as Operator Interface through Port 1

	//print out error and exit if m_pComm doesn't connect
	if(m_pComm->GetLastError() != 0){
		printf("%s", m_pComm->GetLastError());
		printf("\n");
		return 1;
	}

	//set up network callback function
	m_pComm->SetNewNetworkCallBack(NewNetwork, m_pComm);

	//define obstacles
```

```
	DefineObstacles(numObst);
	DefineFormations();
	char keypress;

	//periodic loop to service the communications endpoints
	int i = 0, whosData = 0;

	///Write headers to files
	fprintf(pFile1, "ID	TIME	LAT	LON	ALT");
	fprintf(pFile2, "ID	TIME	LAT	LON	ALT");
	fprintf(pFile3, "LeadTime	TrailTime	2DSeparation	AngleSeparation	AltSeparation");
	while(m_pComm && i == 0)
	{
		m_pComm->RunNetwork();
		int NumNets=m_pComm->GetNumNets();
		LookForAutopilotData(pQ, whosData);
		DoStuff(numObst, NumNets);

		if (kbhit()){
			keypress = getch();			//get commands via keypress
			switch(keypress)
			{
				case 'x':
					i = 1;
					printf("\n");
					AutopilotLoopCmd_t altCom;
					altCom.Loop=1;
					altCom.Control=0;		//turn ap_loop_cmd back
to off
					altCom.Value=current_telemetry[lead_index].Altitude;
					m_pComm->SendAutopilotLoopControlPacket(trailID,
&(altCom));//send the command
					AutopilotLoopCmd_t speedCom;
					speedCom.Loop = 0;		//command a dynamic
pressure
					speedCom.Control = 0;	//turn ap_loop_cmd back to off
					speedCom.Value = (200);//assign the commanded value
					m_pComm->SendAutopilotLoopControlPacket(trailID,
&(speedCom));//send the command
					/*This resets the commands to "auto".  There is sometimes an
issue
					with a Piccolo not working the next time after the code has
been run.
					Usually, on the commands page of the operator interface,
changing commands to
					Off then On will fix it.  Make sure you press 'x' to exit
program and don't just close
					the window, or else commands will not be reset to "auto"
					*/
					//close files
					fclose (pFile1);
					fclose (pFile2);
```

```
                fclose (pFile3);
                break;
        case 't':                       //print telemetry data for selected
network
                data=0;
                break;
        case 'f':                       //print formation data for selected
network
                data=1;
                break;
        case 'c':                       //print and change formation
characteristics
                data=2;
                break;
        case 'm':
                data=3;
                break;
        case '1':                       //print telemetry data for first
Network
                whosData = 0;
                break;
        case '2':                       //print telemetry data for second
Network
                whosData = 1;
                break;
        case '3':                       //print telemetry data for third
Network
                whosData = 2;
                break;
        case '4':                       //print telemetry data for fourth
Network
                whosData = 3;
                break;
        case '5':                       //print telemetry data for fifth
Network
                whosData = 4;
                break;
        case '6':                       //print telemetry data for sixth
Network
                whosData = 5;
                break;
        case '7':                       //print telemetry data for seventh
Network
                whosData = 6;
                break;
        case '8':                       //print telemetry data for eighth
Network
                whosData = 7;
                break;
        case '9':                       //print telemetry data for ninth
Network
                whosData = 8;
                break;
```

```
                                        case '0':                       //print telemetry data for tenth
Network
                                                whosData = 9;
                                                break;
                                }
                        }
                        //delay to create periodic call, as specified by "Index" in the SDK documentation
                        Sleep(10);
        }

        return 0;
}
//Main


/*
        my_types.h
        This file defines the telemetry, control, and location types
        Made a separate file to reduce size of Piccolo.cpp File

 */


#ifndef _MY_TYPES_H
#define _MY_TYPES_H
#include<windows.h>
#include"CommManager.h"


typedef struct
{
        double Longitude;       //from LLA data: Telemtry packet
        double Latitude;  //from LLA data: Telemtry packet
        double East;            //calculated from LLA data using lla2enu class
        double North;           //calculated from LLA data using lla2enu class
        double Up;              //calculated from LLA data using lla2enu class
        float Altitude;         //from LLA data: Telemtry packet
        float Velocity;         //from GPS.Speed: Telemetry packet
        float Beta;             //angle between velocity and direction of nose of plane horizantally
        float Direction;
        int Hours;
        int Minutes;
        float Seconds;
        UInt32 Time;

} telemetry;

//data structure to hold control packet data
typedef struct
{
        float Heading;          //from Yaw reading: Control Data packet
        float BankAngle; //from Roll: Control Data packet
        float RollRate;         //from Roll Rate: Control Data packet
```

```
        float PitchRate;    //from Pitch Rate: Control Data packet
        float YawRate;              //from Yaw Rate: Control Data packet
        float Aileron;
        float Elevator;
        float Throttle;
        float Rudder;
        float AirSpeed;
        int Hours;
        int Minutes;
        float Seconds;
        float Pdynamic;
} control;

//data structure to hold location of obstacles
typedef struct
{
        double Lat;
        double Lon;
        double Alt;
} location;


typedef struct
{
        int radial_sep;
        int z_sep;
        int bearing;
} formation;
                                                #endif
```

Appendix C – Matlab M-Files

```
clc,close all
clear all

%Analysis of Hardware in the Loop Sim with Flight Test
%Orbit waypoints. Variations in Speed and Convergence

if exist('Alt0x5Bm0x5D') == 0
   load 23Mar_562_mod_with_ENU.mat
   disp('File Loading')
end

%Read in Raw flight data from ".mat" file, and build custom Arrays
[Clock] = [Clock0x5Bms0x5D/1000,Day,Hours,Minutes,Seconds];
[Autopilot] = [rad2deg(Lat0x5Brad0x5D),...
   rad2deg(Lon0x5Brad0x5D),...
   Height0x5Bm0x5D...
   TAS0x5Bm0x2Fs0x5D...
   Track_Cmd
   ];

[Heading] = [rad2deg(Direction0x5Brad0x5D)];

[Autopilot_Flight] = [Clock,Autopilot];
BaseX=503000;
BaseY=-4884700;
BaseZ=4057800;

%Waypoint Locations
WP_latitude = [39.774835;
            39.774932;
            39.775029;

            39.775029;
            39.775029;
            39.775029;
            39.775029;
            39.776001;
            39.778040;
            39.779206;
            39.779789;
            39.779886;
            39.779886;
            39.779983;
            39.779983;
            39.779983;
            39.779983;
            39.778623;
            39.777166;
            39.775709;
            39.774835];
```

```
WP_longitude = [-84.097530;
                -84.099473;
                -84.101319;
                -84.103456;
                -84.105884;
                -84.108312;
                -84.110255;
                -84.111809;
                -84.111906;
                -84.111032;
                -84.109187;
                -84.107341;
                -84.105495;
                -84.103358;
                -84.101027;
                -84.098987;
                -84.097142;
                -84.095879;
                -84.095296;
                -84.095782;
                -84.097530];
WP_Altitude(1:21) = 400;
WP_Altitude=WP_Altitude';

lla=[deg2rad(Autopilot_Flight(:,6)) deg2rad(Autopilot_Flight(:,7)) Autopilot_Flight(:,8)];
enu=lla2enu(lla, [BaseX BaseY BaseZ]);
lla_WP=[deg2rad(WP_latitude) deg2rad(WP_longitude) WP_Altitude];
enu_WP=lla2enu(lla_WP, [BaseX BaseY BaseZ]);

%Note enu_WP(:,1), enu_WP(:,2), and enu(:,3) are East, North, Up
%Coordinates of Flight Plan, respectively

%Similarly, enu(:,1), etc are ENU Coordinates of aircraft's actual path

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%HITL Sim with Flight Test TAS(16m/s), Alt(400m), WPs, Gains%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
begin = 867;  %Line # in 'Clock' array
end_at = 8438;

%2D-Plot in LLA
figure('Name',...
   'HITL Sim  TAS(16m/s), Alt(400m), WPs, Gains',...
   'NumberTitle','on')
hold on
plot(Autopilot_Flight(begin:end_at,7),Autopilot_Flight(begin:end_at,6),...
   '--k')
axis equal
xlabel ('Longitude (deg)')
ylabel  ('Latitude (deg)')
title...
   ('HITL Autopilot Simulation #1 : TAS(16m/s), Alt(400m)')
```

```
plot(WP_longitude,WP_latitude,'-ro',...
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[.49 1 .63],...
            'MarkerSize',12);
grid on
axis equal
legend({'UAV Flight Path','Desired Waypoints and FlightPath'});
print -dmeta '1 HITL Autopilot Sim,LLA,2D,Actual'


%2D-Plot in ENU
figure
plot(enu(begin:end_at,1), enu(begin:end_at,2))
hold on
plot(enu_WP(:,1),enu_WP(:,2),'-ro',...
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[.49 1 .63],...
            'MarkerSize',12);
axis square
grid on
xlabel('East (m)');
ylabel('North (m)');

axis equal
grid on
title('HITL Autopilot Simulation #1 : TAS(16mft/s), Alt(400m)');
legend({'UAV Flight Path','Desired Waypoints and FlightPath'});
print -dmeta '2 HITL Autopilot Sim,ENU,2D,Actual'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot 3D Waypoint Orbit Track
figure1 = figure('Name','HITL Sim 3D, Flight Test TAS(16m/s), Alt(400m), WPs,
Gains','NumberTitle','on')
axes1 = axes(...
  'CameraPosition',[-84.13 39.75 2007],...
  'CameraUpVector',[0.1859 0.1775 1.915e+005],...
  'Parent',figure1);
axis(axes1,[-84.11 -84.09 39.77 39.78 -300 -400]);
title(axes1,'HITL Autopilot Simulation #1 : TAS(16m/s), Alt(400m)');
xlabel(axes1,'Longitude (deg)');
ylabel(axes1,'Latitude (deg)');
zlabel(axes1,'Altitude (m)');
grid(axes1,'on');
hold(axes1,'all');
plot3(Autopilot_Flight(begin:end_at,7),...
    Autopilot_Flight(begin:end_at,6),...
    Autopilot_Flight(begin:end_at,8),'Parent',axes1);
grid on
hold on
axis equal
plot3(WP_longitude,WP_latitude,WP_Altitude,'-ro',...
```

```
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[.49 1 .63],...
            'MarkerSize',12);
axis square
legend1 = legend(axes1,...
{'UAV Flight Path','Desired Waypoints,Flight Path, and Altitude (400m)'},...
'Position',[0.2723 0.3165 0.6554 0.1]);
zlim([1000 1400])
print -dmeta '3 HITL Autopilot Sim,LLA,3D,Actual'

%%3D-Plot in ENU Coordinates
figure2 = figure('Name','HITL Sim 3D, Flight Test TAS(16m/s), Alt(400m), WPs,
Gains','NumberTitle','on')
axes2 = axes(...
  'CameraPosition',[0 1000 -200],...
  'CameraUpVector',[0.1859 0.1775 1.915e+004],...
  'Parent',figure2);

plot3(enu(begin:end_at,1), enu(begin:end_at,2), enu(begin:end_at,3),'Parent',axes2);
hold on
plot3(enu_WP(:,1),enu_WP(:,2), enu_WP(:,3),'-ro',...
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[.49 1 .63],...
            'MarkerSize',12);
xlabel(axes2,'East (m)');
ylabel(axes2,'North (m)');
zlabel(axes2,'Up (m');
axis(axes2,[500 2500 1700 2600 -400 -300])
title('HITL Autopilot Simulation #1 : TAS(16m/s), Alt(400m)');
zlim([-400 -300]);
grid on
axis square
legend2 = legend({'UAV Flight Path','Desired Waypoints,Flight Path, and Altitude (400m)'},...
'Position',[0.2723 0.3165 0.6554 0.1]);
print -dmeta '4 HITL Autopilot Sim,ENU,3D,Actual'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%TAS and Altitude versus Time
figure('Name','HITL Sim TAS and Alt, Flight Test TAS(16m/s), Alt(400m), WPs,
Gains','NumberTitle','on')
subplot(2,1,1)
plot(Autopilot_Flight(begin:end_at,1),Autopilot_Flight(begin:end_at,9))
xlabel ('Time (s)')
ylabel  ('Airspeed (ft/sec)')
grid on
subplot(2,1,2)
plot(Autopilot_Flight(begin:end_at,1),Autopilot_Flight(begin:end_at,8),'k')
hold on
xlabel ('Time (s)')
ylabel  ('Altitude (ft)')
plot(time, WP_Altitude(1),'-ro',...
```

```
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[.49 1 .63],...
            'MarkerSize',5);
grid on
avg_TAS_Actual1 = (sum(Autopilot_Flight(begin:end_at,9)))/length(Autopilot_Flight(begin:end_at,9))
print -dmeta '5 HITL Autopilot Sim,TAS Alt,Actual'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%DEVIATIONS AT WAYPOINT CALCULATIONS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

i=1;
for(counter=begin:1:end_at)
   if(Autopilot_Flight(counter,10)~= Autopilot_Flight(counter+1,10))
     indices_at_hitting_waypoints(i)=counter;
     waypoint_index(i)=Autopilot_Flight(counter,10);
     time(i)=Autopilot_Flight(counter,1);
     i=i+1;
   end
end

for(counter=1:length(indices_at_hitting_waypoints))
   for(j=begin:end_at)
      east_deviation(j)=(enu(j,1)-enu_WP(waypoint_index(counter)+1,1));
      north_deviation(j)=(enu(j,2)-enu_WP(waypoint_index(counter)+1,2));
      xy_deviation(j)=sqrt(east_deviation(j)^2+north_deviation(j)^2);
   end
   [junk(counter), index(counter)]=min(xy_deviation(begin:end_at));
   index(counter)=index(counter)+begin;
   alt_deviation_at_waypoints(counter)=((Autopilot_Flight(index(counter),8)-
WP_Altitude(waypoint_index(counter)+1)));
   east_deviation_at_waypoints(counter)=east_deviation(index(counter));
   north_deviation_at_waypoints(counter)=north_deviation(index(counter));
   xy_deviation_at_waypoints(counter)=xy_deviation(index(counter));
end

avg_alt_deviation_at_waypoints = sum(abs(alt_deviation))/length(alt_deviation)
avg_xy_deviation_at_waypoints =
sum(abs(xy_deviation_at_waypoints))/length(xy_deviation_at_waypoints)
maximum_xy_deviation=max(xy_deviation_at_waypoints)
maximum_alt_deviation=max(alt_deviation_at_waypoints)


%XY-Deviation at Waypoints
figure
plot(time, xy_deviation_at_waypoints);
hold on
plot(time, 0,'-ro',...
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[.49 1 .63],...
```

```
            'MarkerSize',5);
xlabel('Time');
ylabel('XY-Deviation at Waypoints (ft)');
axis([Autopilot_Flight(begin) Autopilot_Flight(end_at) 0 300])
grid on
title('HITL Autopilot Simulation #1 : TAS(16m/s) Alt(400m)');
legend({'UAV Flight Path','Desired Waypoints and FlightPath'});
print -dmeta '6 HITL Autopilot Sim,XY-Deviation,Actual'
```

Note that similar codes were used to import and analyze formation data. The names of variables and particular graphs were changed, but the methods were the same.

Appendix D – Flight Test Cards

TASK ID TASK

Pat-1 Single A/C Waypoint Tracking

FLIGHT PHASE	TASK DESCRIPTION
Low Altitude Cruise	Single A/C Waypoint Tracking

FIXED PARAMETERS

Initial Position: Straight and level flight Altitude 400m	Flight Plan: Area B Racetrack	

VARIED PARAMETERS

Speed (m/s)	Track Convergence					
16,20	50,250					

TEST PROCEDURE

PILOT

1. Maintain straight and level flight on straightaway of racetrack pattern
2. Switch to autonomous flight, record data for 3 racetrack loops
3. Change gains, repeat.

TEST ENGINEER/PILOT NOT FLYING

PILOT	DATE	RUN NUMBER

EVALUATION SEGMENT	LONG CHR	LAT/DIR CHR
Waypoint Tracking		

Start Evaluation: Straight and level flight
End Evaluation: Straight and level flight

EVALUATION BASIS

Normal autonomous flight under varying airspeeds and track convergences. Deviation at waypoints will be recorded.

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE
2-D Deviation from Waypoints (m)	0	±20	±60
Deviation in Altitude (m)	0	±10	±50
Deviation in Airspeed (m/s)	0	±1	±3

TASK ID TASK

Pat-2 Single A/C Turning Radius

FLIGHT PHASE	TASK DESCRIPTION	PILOT	DATE	RUN NUMBER
Low Altitude Cruise	Single A/C Turning Radius			

FIXED PARAMETERS

Initial Position: Straight and level flight Altitude 400m	Flight Plan: East-West Straight Line	

EVALUATION SEGMENT	LONG CHR	LAT/DIR CHR
Turning Radius		

Start Evaluation: Straight and level flight
End Evaluation: Straight and level flight

EVALUATION BASIS

Following a straight line path between two waypoints. Studying the minimum turning radius at various airspeeds.

VARIED PARAMETERS

Speed (m/s)	Track Convergence					
12,16,20	50					

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE
Turning Radius (m)	75,125,175	±50	±100
Deviation in Altitude (m)	0	±20	±50
Deviation in Airspeed (m/s)	0	±1	±3

TEST PROCEDURE

PILOT

1. Maintain straight and level flight on straightaway
2. Switch to autonomous flight, record data for 4 turns
3. Change airspeed, repeat

TEST ENGINEER/PILOT NOT FLYING

TASK ID TASK

Pat-3 Formation Flight Time Delay

FLIGHT PHASE	TASK DESCRIPTION	PILOT	DATE	RUN NUMBER
Low Altitude Cruise	Formation Flight Time Delay – Airspeed and Altitude Changes			

FIXED PARAMETERS

Initial Position: Straight and level flight Altitude 400m	Flight Plan: East-West Straight Line	Lead Track Convergence: 50 Lead Altitude: 330m Trail: 50m above Lead

VARIED PARAMETERS

Lead Speed (m/s) 14-20		Lead Altitude (m) 330-380			

EVALUATION SEGMENT	LONG CHR	LAT/DIR CHR
Time Delay		

Start Evaluation: Straight and level flight
End Evaluation: Straight and level flight

EVALUATION BASIS

Flying a straight-line path in formation with C++ code turned ON. Lead will be commanded new airspeeds and altitudes and trail will follow.

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE
Time Delay (s)	3	±2	±5

TEST PROCEDURE

PILOT

1. Maintain straight and level flight on straightaway
2. Switch to autonomous flight, command new lead airspeed
3. On next straightaway, command new altitude

TEST ENGINEER/PILOT NOT FLYING

TASK ID TASK

Pat-4 Formation Flight Tracking

FLIGHT PHASE	TASK DESCRIPTION	PILOT	DATE	RUN NUMBER
Low Altitude Cruise	Formation Flight Tracking			

FIXED PARAMETERS

Initial Position: Straight and level flight Altitude 400m	Flight Plan: Area B Racetrack	Lead Altitude: 350m Trail: 50m above Lead

VARIED PARAMETERS

Lead Speed (m/s) 16,20		Lead Track Convergence 50,250			

TEST PROCEDURE

PILOT

1. Maintain straight and level flight on straightaway
2. Run C++ code
3. With trail flying same Area B Racetrack as lead, 100-250m behind in straightaway, similar headings, switch trail to autonomous
3. Fly 3 passes
4. Vary airspeed or track convergence
5. Repeat Steps 3-4
6. Switch back to manual control, press 'X' to exit C++ code.

TEST ENGINEER/PILOT NOT FLYING

EVALUATION SEGMENT	LONG CHR	LAT/DIR CHR
Formation Flight Tracking		

Start Evaluation: Straight and level flight
End Evaluation: Straight and level flight

EVALUATION BASIS

Formation flying with simulated lead aircraft. Vary lead aircraft's speed and track convergence. Trail flying 50m above lead.

PERFORMANCE STANDARDS	TARGET	DESIRED	ADEQUATE
2-D Formation Separation (m)	60	±20	±50

Appendix E – Flight Test Results

This Appendix includes the results of flight testing that was conducted after the defense of this thesis. The results were appended to the document after it was complete.

Several flight tests were conducted. Due to time constraints, no single aircraft performance characterization maneuvers were performed. Instead of flying a simulated lead aircraft with trail aircraft in the sky, it was deemed that a simulated trail aircraft would be safer for the first time using the formation flight controller in flight tests. The setup was identical to that described in Chapter V, except the trail was the simulated aircraft and the lead was flown on "Manual" mode by the R/C pilot. Flying autonomously was impossible on this day due to issues with the pitot-static tube. The airspeed of the airborne aircraft fluctuated between proper readings and zero. Without an accurate measure of TAS, autonomous flight would be disastrous. The flight tests that were conducted included airspeed and altitude changes, and a formation flight around the Area B racetrack. The altitude changes are shown in Figure 95.

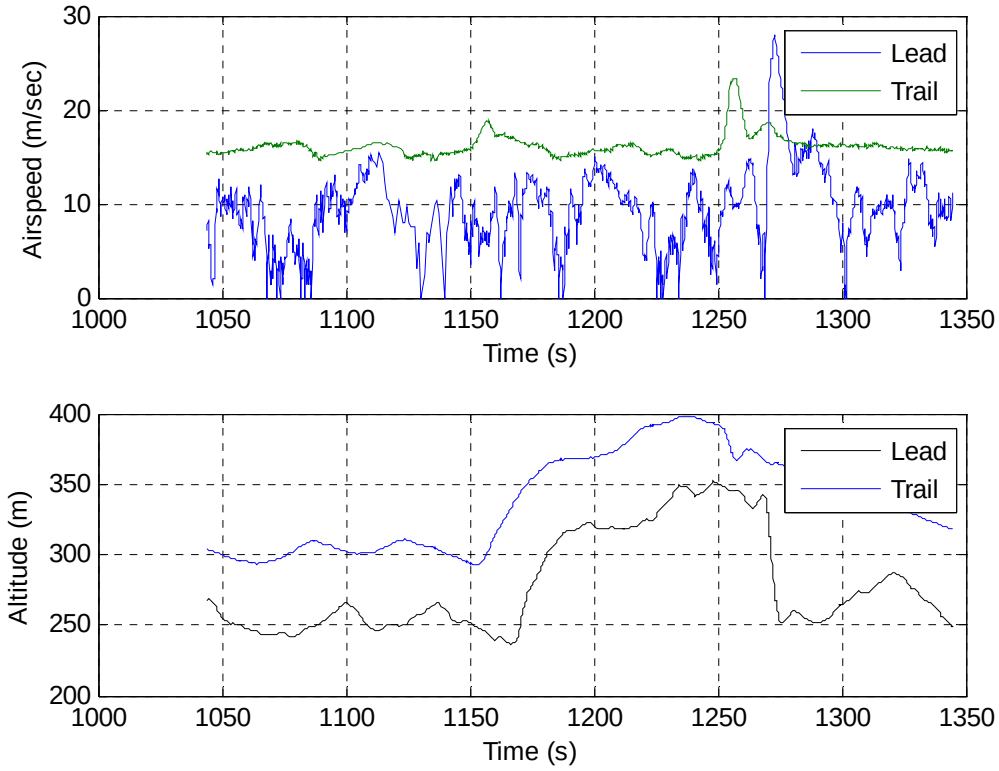

Figure 95 - Flight Test: Altitude Change

The lead aircraft was flown by the R/C pilot, so all maneuvers will not be as fluid as autonomous flight may have achieved. Flying in a racetrack pattern, the pilot raised the altitude of the lead aircraft while maintaining airspeed. As Figure 95 shows, the lead aircraft climbed approximately 70m. The time delay was even more significant that HITL simulations predicted, with the trail not beginning its climb until 12 seconds after the lead. This error can be attributed to the problems with the pitot-static tube in the lead aircraft. Fluctuations in lead TAS sent equally fluctuating commands to the trail. Figure 96 shows an airspeed change in the lead. The time delay here is impossible to tell because of the lead aircraft's TAS fluctuation. Altitude was also not properly held constant during this flight test.

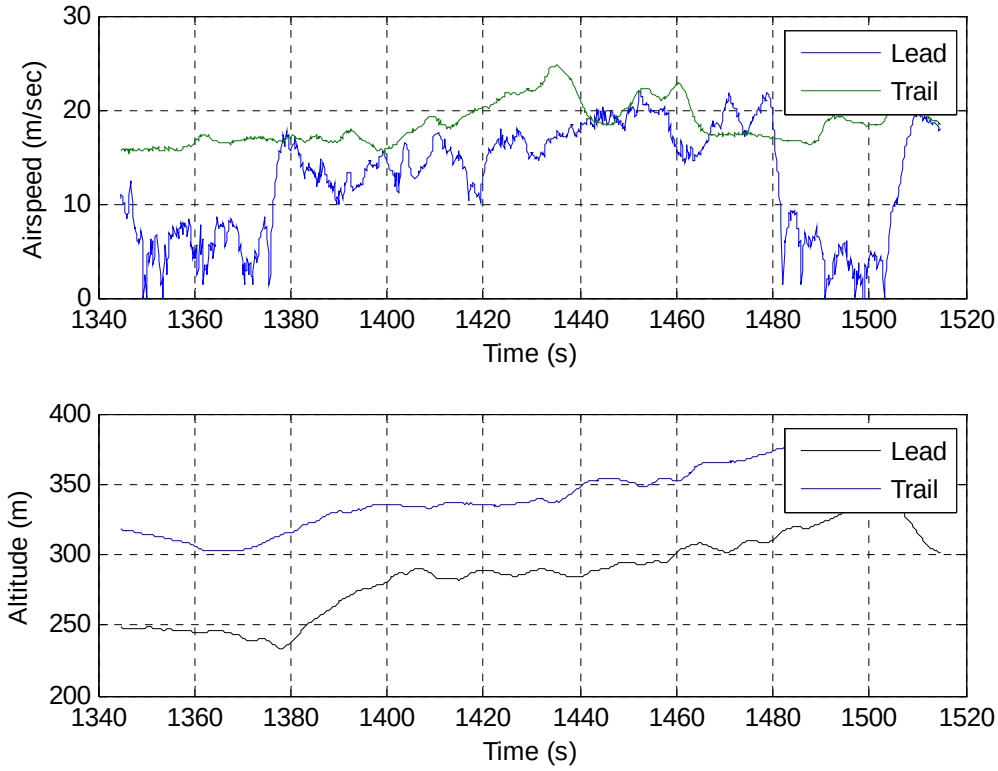

Figure 96 - Flight Test: Airspeed Change

Figure 97 and Figure 98 show the results of the formation flight around a racetrack. Again, due to fluctuating lead airspeeds, the trail wasn't able to maintain any semblance of satisfactory formation flight. The desired waypoints moved properly along with the lead aircraft, but as Figure 98 shows, the trail airspeed never really changes because it is constantly seeing extreme fluctuations in lead airspeed. It is being commanded such extreme airspeed changes that it doesn't have any time to react. The exact formation separations were not calculated, because proper flight testing will be conducted in later research after the issues described above are fixed. The important thing to draw from these flight tests are that the formation flight controller functioned properly. One other thing to consider when analyzing Figure 97 and Figure 98 is that the lead aircraft was

being flown manually, not autonomously. Therefore, the lead aircraft wasn't intending to fly the exact track pictured. The waypoints are merely guidelines given to the R/C pilot to give him an idea of the desired racetrack.

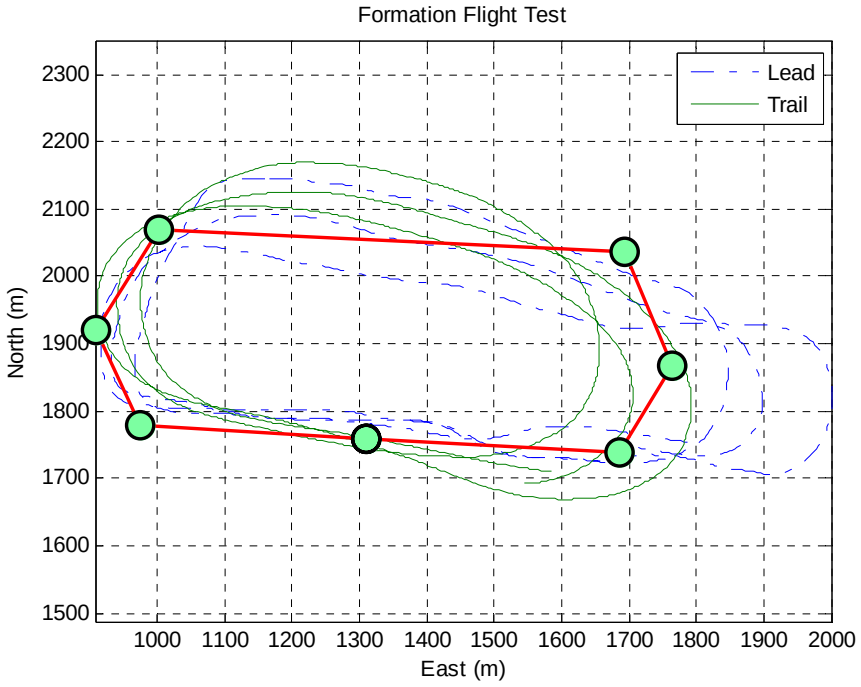

Figure 97 - Flight Test: Formation Flight

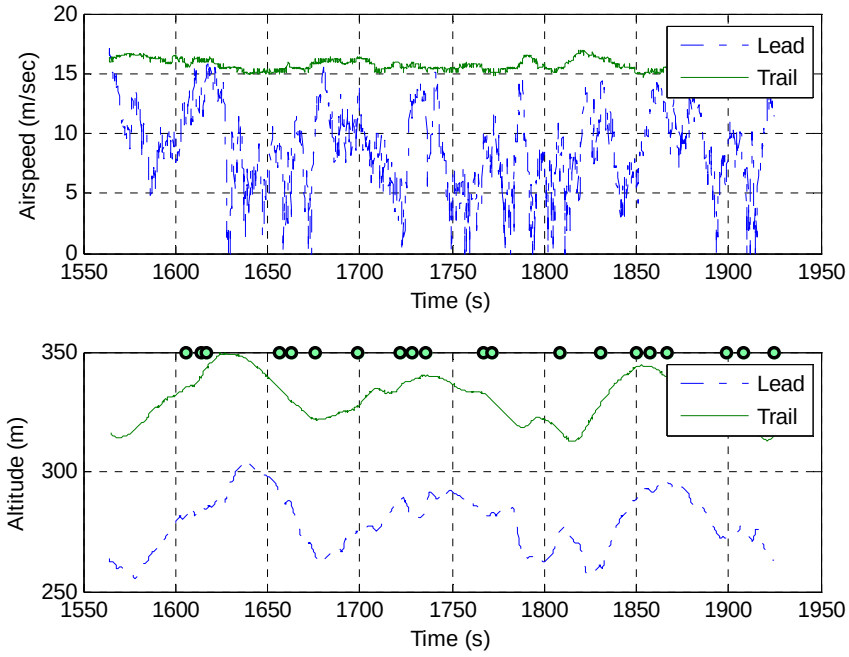

Figure 98 - Flight Test: Formation Flight

Bibliography

Baleri, Giri. "Datum Transformations of NAV420 Refernce Frames. Crossbow Technology." Retrieved January 27, 2006 from http://www.xbow.com/Support/Support_pdf_files/NAV420AppNote.pdf

Bayraktar, S., Fainekos, G. E., Pappas, G. J. "Experimental Cooperative Control of Fixed-Wing Unmanned Aerial Vehicles." *43rd IEEE Conference on Decision and Control*. Bahamas. December 2004.

Borys, D. N., Colgren, R. "Advances in Intelligent Autopilot Systems for Unmanned Aerial Vehicles." *AIAA Guidance, Navigation, and Control Conference and Exhibit*. August 2005. (AIAA-2005-6482-487).

"Cloud Cap Technology – What's New." Retrieved June 3, 2006 from http://www.cloudcaptech.com/whatsnew.htm#stanag. 2006

Dugan, J. M. *Situational Awareness and Synthetic Vision for Unmanned Aerial Vehicle Flight Testing*. MS Thesis, AFIT/GAE/ENY/06-J03. School of Engineering and Management, Air Force Institute of Technology (AFIT), Wright-Patterson AFB, OH. June 2006.

Frew, E., Xiao, X., Spry, S., McGee, T., Kim, Z., Tisdale, J., Sengupta, R., Hendrick, K.J. "Flight Demonstrations of Self-directed Collaborative Navigation of Small Unmanned Aircraft." *Proceedings of the 2004 IEEE Aerospace Conference*. Big Sky, MT. March 2004.

Four Cycle Engine Owner's Instruction Manual. Japan. O.S. Engines Mfg. Co. 2000.

Hanson, C. E., Ryan, J., Allen, M.J., Jacobson, S.R. "An Overview of Flight Test Results for a Formation Flight Autopilot." NASA/TM-2002-210729. August 2002.

Jodeh, Nidal M. *Development of Autonomous Unmanned Aerial Vehicle Research Platform: Modeling, Simulating, and Flight Testing*. MS Thesis, AFIT/GAE/ENY/06-M18. School of Engineering and Management, Air Force Institute of Technology (AFIT), Wright-Patterson AFB, OH. March 2006.

King, Ellis T. *Distributed Coordination and Control Experiments on a Multi-UAV Testbed*. MS Thesis, Massachusetts Institute of Technology (MIT). September 2004.

Lozier, M., Jodeh, N. "Advanced Navigation Technology Demonstrator Initial Flight Tests: Safety Review Board." October 2005.

Nelson, Robert C. *Flight Stability and Automatic Control* (Second Edition). Madison, WI. McGraw-Hill, 1998.

Osteroos, Ryan K. *Full Capability Formation Flight Control*. MS Thesis, AFIT/GAE/ENY/05-M16. School of Engineering and Management, Air Force Institute of Technology (AFIT), Wright-Patterson AFB, OH. February 2005.

Robinson, B. *An Investigation into Robust Wind Correction Algorithms for Off-the-Shelf Unmanned Aerial Vehicle Autopilots.* MS Thesis, AFIT/GAE/ENY/06-J14. School of Engineering and Management, Air Force Institute of Technology (AFIT), Wright-Patterson AFB, OH. June 2006.

Rogers, David F. "Turn Performance." Retrieved on April 14, 2005 from http://web.usna.navy.mil/~dfr/flying/turnwide.pdf. 2001.

Ross, Steven M. *Formation Flight Control for Aerial Refueling.* MS Thesis, AFIT/GAE/ENY/06-M35. School of Engineering and Management, Air Force Institute of Technology (AFIT), Wright-Patterson AFB, OH. March 2006.

Ryan, A., Zennaro, M., Howell, A., Sengupta, R., Hedrick, J.K. "An Overview of Emerging Results in Cooperative UAV Control." 43rd IEEE Conference on Decision and Control. Bahamas. December 2004.

Stevens, B. L., Lewis, F. L. *Aircraft Control and Simulation* (Second Edition). Joboken, NJ. John Wiley & Sons, Inc., 2003.

Teo, R., Jung Soon J., Tomlin, C.J. "Automated Multiple UAV Flight – The Stanford DragonFly UAV Program." *43rd IEEE Conference on Decision and Control.* Bahamas. December 2004.

Tin, Chung. *Robust Multi-UAV Planning in Dynamic and Uncertain Environments.* MS Thesis, Massachusetts Institute of Technology (MIT). August 2004.

U-Blox AG, Switzerland. Retrieved on April 14, 2005, from http://www.u-blox.com/products/tim_lp.html

Vaglienti, B., Hoag, R., Niculescu, M. *Piccolo System Users Guide.* Hood River OR. Cloud Cap Technology. 18 April 2005.

Vaglienti, B., Niculescu, M. *Hardware in the Loop Simulator for the Piccolo Avionics.* Hood River OR. Cloud Cap Technology. 18 April 2005.

Vita

Ensign Patrick A. McCarthy graduated from Thomas Jefferson High School for Science and Technology in Alexandria, Virginia. He entered undergraduate studies at the University of Notre Dame in South Bend, Indiana, where he graduated with a Bachelor of Science degree in Aerospace Engineering in May 2005. He was commissioned through the NROTC Unit at the University of Notre Dame, where he was selected into the aviation community with a pilot designation.

His first assignment was at Wright-Patterson AFB as a student in the Graduate School of Engineering and Management, Air Force Institute of Technology, in June 2005. Upon graduation, he will report to Pensacola, Florida for pilot training.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
13-06-2006	Master's Thesis	September 2005 – June 2006

4. TITLE AND SUBTITLE

Characterization of UAV Performance and Development of a Formation Flight Controller for Multiple Small UAVs

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

McCarthy, Patrick, A, Ensign, USN

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GAE/ENY/06-J08

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/VA
2219 8th St., WPAFB, OH, 45433
Lt Col Lawrence Leny
(937) 255-6500

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Air Force Institute of Technology's (AFIT) Advanced Navigation Technology (ANT) Center has recently delved into the research topic of small Unmanned Aerial Vehicles (UAV). One area of particular interest is using multiple small UAVs cooperatively to improve mission efficiency, as well as perform missions that couldn't be performed using vehicles independently. However, many of these missions require that the UAVs operate in close proximity of each other. This research lays the foundation required to use the ANT Center's UAVs for multi-vehicle missions (e.g. cooperatively) by accomplishing two major goals. First, it develops test procedures that can be used to characterize the tracking performance of a small UAV being controlled by a waypoint guided autopilot. This defines the size of the safety zones that must be maintained around each vehicle to ensure no collisions, assuming no, as yet unspecified, collision avoidance algorithm is being implemented. Secondly, a formation flight algorithm is developed that can be used to guide UAVs relative to each other using a waypoint guided autopilot. This is done by dynamically changing the waypoints. Such an approach gives a "wrap-around" method of cooperatively controlling UAVs that can only be guided waypoint-to-waypoint. For both components of this research, tests were conducted using a hardware-in-the-loop (HITL) simulation before validating through flight testing. This report, along with legacy documentation and procedures, furthers the UAV test bed at AFIT and establishes methods for simulating, visualizing, and flight testing multiple UAVs during formation/cooperative flight.

15. SUBJECT TERMS

UAV, Formation Flight Controller, Cooperative Control, UAS, Waypoint Guided Autopilot

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Paul A. Blue, Major, USAF (ENY)
U	**U**	**U**	**UU**	165	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565 x4714 (Paul.Blue@afit.edu)